P2: Dependency Parsing

Assignment overview, Hints and Tips

Natural Language Understanding

Interuniversity Master's Degree in Artificial Intelligence
Academic Year 2025-2026

UNIVERSIDADE DA CORUÑA

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

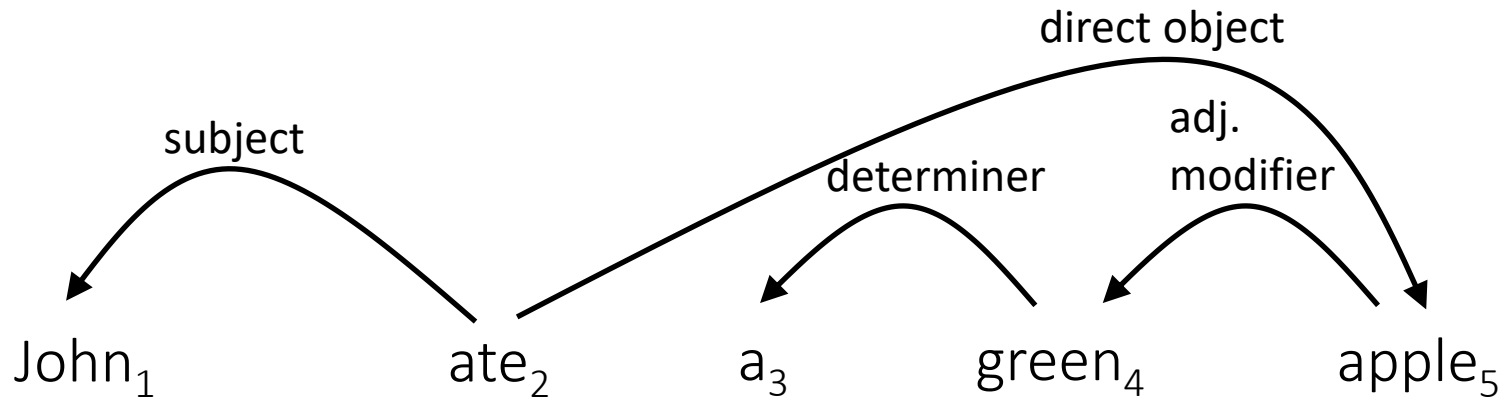
Universidade Vigo

Objectives

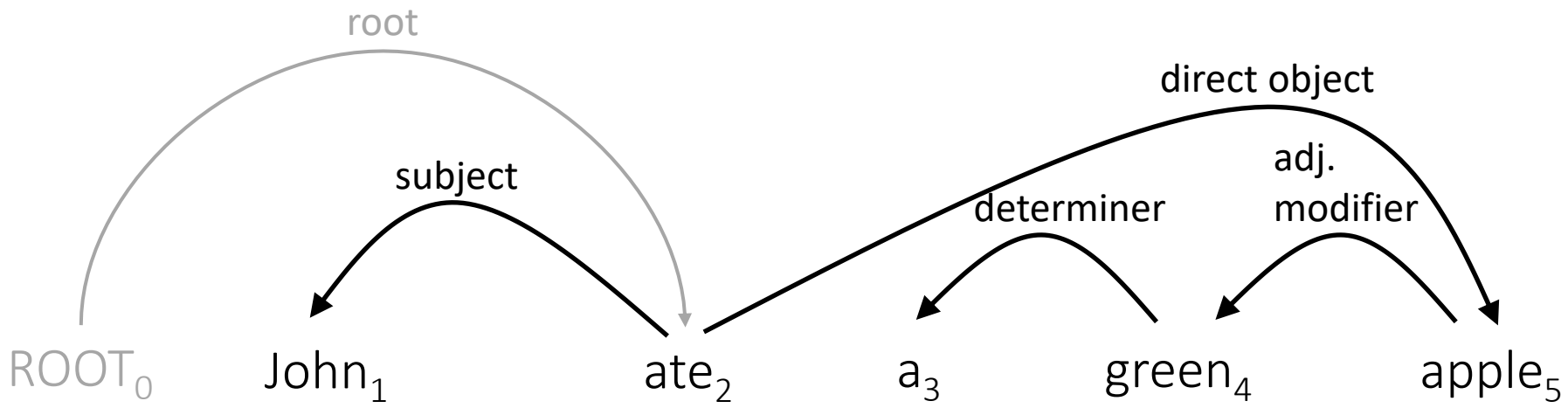
- Work and process successfully Universal Dependency (UD) datasets for dependency parsing
- Build and train your own neural parser
  - Implement the transition-based algorithm: arc-eager parsing algorithm
  - Output: the dependency syntactic structure (in CoNLLU format)
- Compare and discuss variations of models and their performance, reporting the standard metrics for parsing (LAS, UAS)

Dependency graph

Informally, “who do what to whom”



Dependency graph



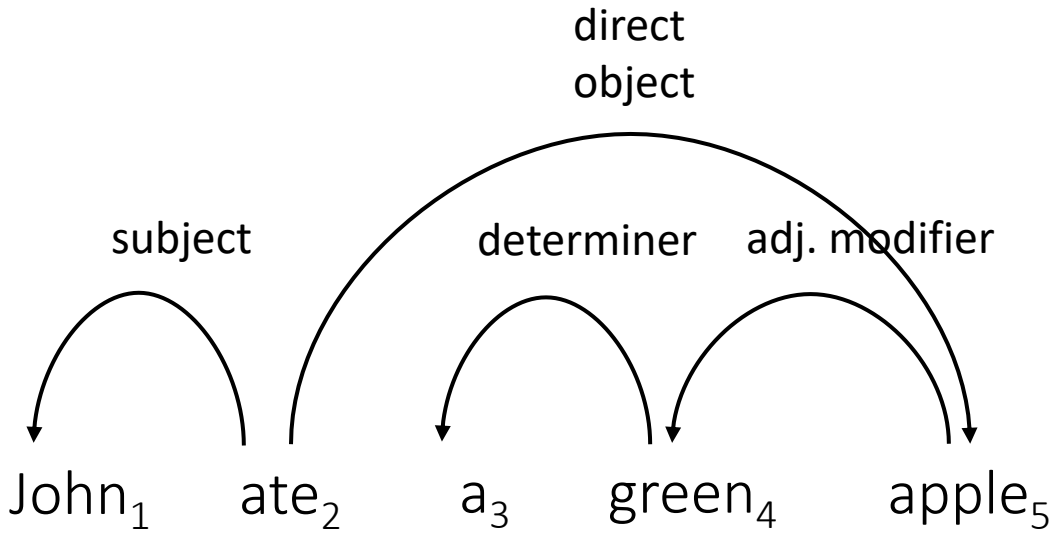
Each token has exactly one head (e.g. ‘ate’ is the head of ‘John’)

Each token can have 0 or more dependents (e.g. ‘ate’ has two dependents: ‘John’ and ‘Apple’, while ‘a’ has none).

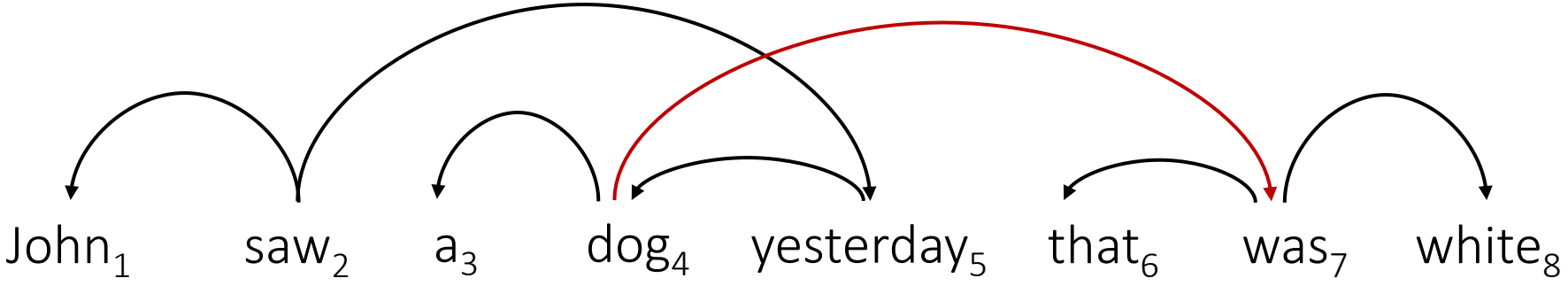
A token with ID 0, called *dummy root*, is usually included to mark the real sentence root.

Dependency graph

Projective graphs can be drawn with no crossing edges…



…while *non-projective graphs* cannot.



CoNLL-U Format REVISITED

- Sentences consist of one or more word lines, and word lines contain the following fields:
  - ID: Word index, integer starting at 1 for each new sentence; may be a range for multiword tokens; may be a decimal number for empty nodes (decimal numbers can be lower than 1 but must be greater than 0).
  - FORM: Word form or punctuation symbol.
  - LEMMA: Lemma or stem of word form.
  - UPOS: Universal part-of-speech tag.
  - XPOS: Language-specific part-of-speech tag; underscore if not available.
  - FEATS: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.
  - HEAD: Head of the current word, which is either a value of ID or zero (0).
  - DEPREL: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
  - DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.
  - MISC: Any other annotation.

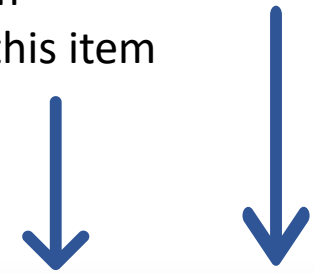
https://universaldependencies.org/format.html

CoNLL-U Tree Viewer: https://urd2.let.rug.nl/~kleiweg/conllu/

Example CoNLLU

Dependency relationship (label) for the arc

ID as head for this item
=> arc between the head and this item



ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
1	Creative	Creative	PROPN	SP	_	3	nmod	_	_
2	Commons	Commons	PROPN	SP	_	1	flat	_	_
3	Corporation	corporation	NOUN	S	Number=Sing	8	nsubj	_	_
4	is	be	AUX	V	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	8	cop	_	_
5	not	not	PART	PART	Polarity=Neg	8	advmod	_	_
6	a	a	DET	RI	Definite=Ind|Number=Sing|PronType=Art	8	det	_	_
7	law	law	NOUN	S	Number=Sing	8	nmod	_	_
8	firm	firm	NOUN	S	Number=Sing	0	root	_	_
9	and	and	CCONJ	CC	_	12	cc	_	_
10	does	do	AUX	VM	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	12	aux	_	_
11	not	not	PART	PART	Polarity=Neg	12	advmod	_	_
12	provide	provide	VERB	V	Mood=Ind|Number=Plur|Tense=Pres|VerbForm=Fin	8	conj	_	_
13	legal	legal	ADJ	A	Degree=Pos	14	amod	_	_
14	services	service	NOUN	S	Number=Plur	12	obj	_	SpaceAfter=No
15	.	.	PUNCT	FS	_	8	punct	_	_

Preparing CoNLLU data

```
1       Creative        Creative        PROPN   SP      _       3       nsubj   _       _
2       Commons Commons PROPN   SP      _       1       flat    _       _
3       provides        provide VERB    V       Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin   0       root    _
        _
4       this    this    DET     DD      Number=Sing|PronType=Dem        5       det     _       _
5       information     information     NOUN    S       Number=Sing     3       obj     _       _
6       on      on      ADP     E       _       13      case    _       _
7       an      a       DET     RI      Definite=Ind|Number=Sing|PronType=Art   13      det     _       _
8       "       "       PUNCT   FB      _       11      punct   _       _
9       as      as      ADP     E       _       11      mark    _       _
10      -       -       PUNCT   FF      _       9       punct   _       _
11      is      be      VERB    V       Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin   13      amod    _       _
12      "       "       PUNCT   FB      _       11      punct   _       _
13      basis   basis   NOUN    S       Gender=Masc|Number=Sing 3       obl     _       _
14      .       .       PUNCT   FS      _       3       punct   _       _
```

```
[0, 'ROOT', 'ROOT', '_', '_', '_', '_', '_', '_', '_']
[1, 'Creative', 'Creative', 'PROPN', 'SP', '_', 3, 'nsubj', '_', '_']
[2, 'Commons', 'Commons', 'PROPN', 'SP', '_', 1, 'flat', '_', '_']
[3, 'provides', 'provide', 'VERB', 'V', 'Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin', 0, 'root', '_', '_']
[4, 'this', 'this', 'DET', 'DD', 'Number=Sing|PronType=Dem', 5, 'det', '_', '_']
[5, 'information', 'information', 'NOUN', 'S', 'Number=Sing', 3, 'obj', '_', '_']
[6, 'on', 'on', 'ADP', 'E', '_', 13, 'case', '_', '_']
[7, 'an', 'a', 'DET', 'RI', 'Definite=Ind|Number=Sing|PronType=Art', 13, 'det', '_', '_']
[8, '"', '"', 'PUNCT', 'FB', '_', 11, 'punct', '_', '_']
[9, 'as', 'as', 'ADP', 'E', '_', 11, 'mark', '_', '_']
[10, '-', '-', 'PUNCT', 'FF', '_', 9, 'punct', '_', '_']
[11, 'is', 'be', 'VERB', 'V', 'Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin', 13, 'amod', '_', '_']
[12, '"', '"', 'PUNCT', 'FB', '_', 11, 'punct', '_', '_']
[13, 'basis', 'basis', 'NOUN', 'S', 'Gender=Masc|Number=Sing', 3, 'obl', '_', '_']
[14, '.', '.', 'PUNCT', 'FS', '_', 3, 'punct', '_', '_']
```

- We are giving you a class ConlluReader (conllu_reader.py) to reading and processing files in CoNLLU format. It includes:
  - Methods for the conversion of graphs into CoNLLU formatted strings, and viceversa.
  - A method for filtering non-projective graphs.
- You have to prepare dictionaries for columns FORM, UPOS and DEPREL
  - To convert from text/label to a numeric value and viceversa

CoNLLU tokens

We are also providing a Token class (conllu_token.py) to store the information of CoNLLU tokens, with an example of use.

```
class Token():
    """
    A class to encode a line from a CoNLLU file.

    Attributes:
    - id (int): Token ID in the sentence.
    - form (str): The form or orthography of the word.
    - lemma (str): The lemma or base form of the word.
    - upos (str): Universal part-of-speech tag.
    - cpos (str): Language-specific part-of-speech tag; not used in this assignment.
    - feats (str): List of morphological features from the universal feature inventory or language-specific extension; separated by '|'.
    - head (int): Head of the current token, which is either a value of ID or zero ('0').
    - dep (str): Universal dependency relation to the HEAD.
    - deps (str): Enhanced dependency graph in the form of a list of head-deprel pairs.
    - misc (str): Any other annotation.

    Methods:
    - get_fields_list(): Returns a list of all the attribute values of the token, in the same order requited by the conllu format

    # Example usage:

    token_example = Token(1, "Distribution", "distribution", "NOUN", "S", "Number=Sing", 7, "nsubj")
    """
```

Transition-based dependency parsers

Formally represented as tuples (C, T, c_s, C_t):

- C: the set of states.
- T: the set of transitions, a partial function $C \rightarrow C$.
- c_s: the initial state.
- C_t: the set of posible final states.

You will see transition-based parsers in theory class.

In this lab you will implement one of them: the arc-eager algorithm

File Reading and Writing

It is already implemented in the file *conllu_reader.py*, along with other materials for this practice. You shouldn't need to add anything else! Usage examples are included at the end of the file itself.

```
class ConlluReader():
    """
    A class for reading and processing CoNLLU format files. CoNLLU (Conference on Natural Language
    Learning Universal Dependencies) format is widely used for linguistic annotation, especially in
    dependency parsing. This class provides methods to convert between tree structures (as a list of
    Token objects) and CoNLLU formatted strings, as well as to read CoNLLU files.

    Methods:
        tree2conllustr(tree: list[Token]) -> str:
            Converts a tree structure (list of Token objects) into a CoNLLU formatted string.

        conllustr2tree(conllustr: str, inference: bool = True) -> list[Token]:
            Converts a CoNLLU formatted string into a list of Token objects representing a tree.

        read_conllu_file(path: str, inference: bool = False) -> list:
            Reads a CoNLLU file and converts it into a list of tree structures.

        write_conllu_file(path: str, trees: list['Token']):
            Writes a list of tree structures to a file in CoNLLU format.

        remove_non_projective_trees(trees: list['Token']) -> list:
            Filters out non-projective trees from a list of tree structures.

    Note: Example usage is provided at the end of the file.
    """
```

Arc-eager algorithm

The most common way to represent a state in a transition-based model is with a triplet in the form of (σ, β, A):

σ: A stack, which stores partially processed words. It has a Last In, First Out (LIFO) structure.

β: A buffer, which stores the words that still need to be read. It has a First In, First Out (FIFO) structure.

A: The set of arcs in the tree that have already been created.

To transition from one state c_i to another state c_{i+1}, transitions are applied. This process is repeated until reaching a state referred to as *the final state*

Arc-eager algorithm

We are giving to you a class State (state.py) to implement the states of the arc-eager algorithm. An example of use is included.

```
class State(object):
    """
    Class to represent a parsing state in dependency parsing.

    Attributes:
        S (list['Token']): A stack holding tokens that are
                           currently being processed.
        B (list['Token']): A buffer holding tokens that are yet
                           to be processed.
        A (set[tuple]): A set of arcs of the form (head_id, dependency_label, dependent_id)
                        created during parsing, representing the dependencies.

    The class is used in dependency parsing algorithms to maintain the state of the parsing
    process, including which tokens are being considered and the relationships formed between tokens.
    """
```

Arc-eager algorithm

Initial state:

σ: Initially, it only contains the ROOT node, i.e., σ = [0]

β: Initially, it contains the complete sequence used as input, i.e., β=[1,...,N]

A: Initialized as empty, no arcs have been created yet, i.e., A= {}

Final state: any state in which β = [] (it doesn't matter what content is in σ). This varies depending on the transition algorithm we use (you will see this in more detail in the lectures).

Arc-eager algorithm - Transitions

LEFT-ARC$_l$ – Creates an arc (with label l) from the first word in the buffer to the word currently at the top of the stack. Then, it removes the word from the top of the stack.

RIGHT-ARC$_l$ – Creates an arc (with label l) from the word at the top of the stack to the first word in the buffer. Then, it moves the first word from the buffer to the top of the stack.

REDUCE – Removes the word from the top of the stack (important! Applying this transition implies that we have already established all its relationships).

SHIFT – Moves the first word from the buffer to the top of the stack.

We are providing a Transition class (algorithm.py), with an usage example at the end of the file.

Arc-eager algorithm

Transitions as a deductive system

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$

Preconditions for each transition

LEFT-ARC = $\neg[i = 0]$

$\neg\exists k \exists l\ [(k, l, i) \in A]$

RIGHT-ARC = $\neg\exists k \exists l\ [(k, l, j) \in A]$

REDUCE = $\exists k \exists l [(k, l, i) \in A]$

Supervised model for dependency parsing

As is customary, we will have the training, development, and test sets.

The goal is to train a model that, given a sentence, can generate the sequence of transitions that allows us to recover the syntactic structure of the sentence.

Training

We have access to the reference tree, also often referred as the gold tree.

We derive the sequence of transitions that would allow us to reconstruct the tree.

We start in an initial state. In each state, we select the next transition that is valid and correct.

We update the state based on the selected transition.

We repeat the process until we reach a final state.

This is commonly referred to as the *oracle implementation*.

Each pair (state, transition) is a training sample. The state is the input to the model and the transition will be the output.

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

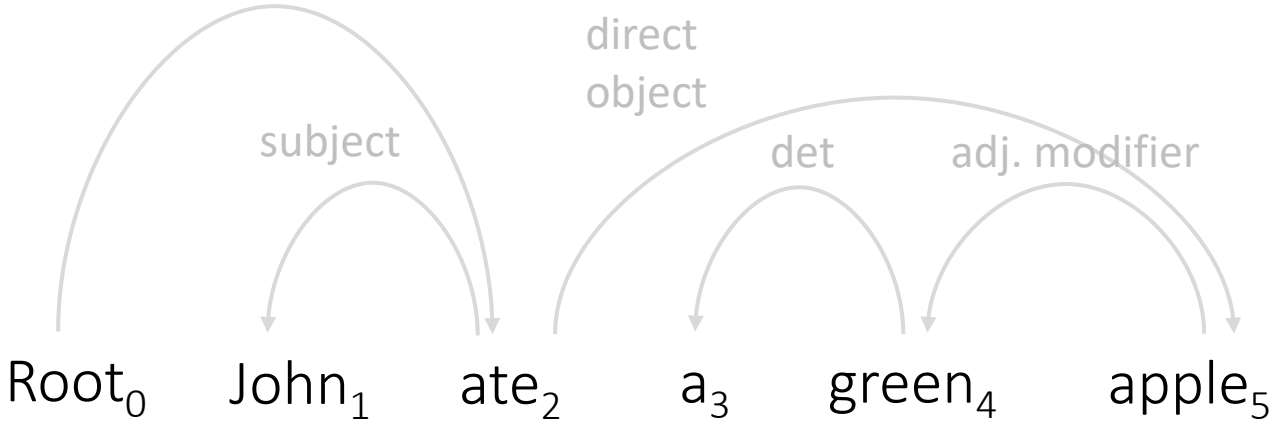
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]		

At each step, we analyze and select the transition that is valid (meets the preconditions) and correct (its application contributes to obtaining the target tree).

LEFT-ARC? It's not valid, the top of the stack is the ROOT (i=0).

RIGHT-ARC? Valid, but incorrect (there is no arc ROOT -> John in the gold tree).

REDUCE? Not valid, ROOT is not dependent on any word yet (specifically, ROOT can never be reduced).

SHIFT? Valid and correct.

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

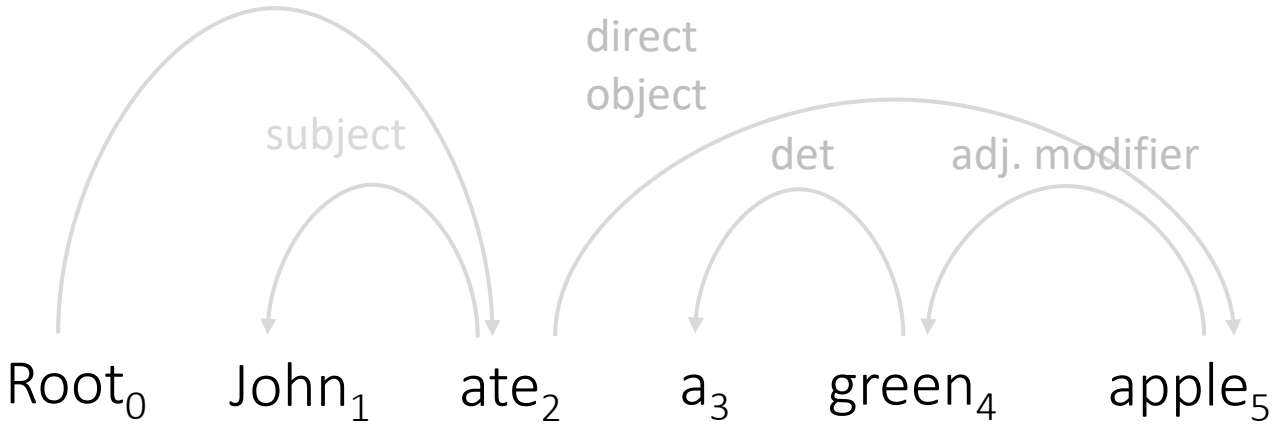
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]		

LEFT-ARC? Valid and correct (there is an arc between $John_1$ <- ate_2)

RIGHT-ARC? Valid, but not correct (there is no arc $John_1$ -> ate_2)

REDUCE? Not valid: $John_1$ has not been assigned a parent yet.

SHIFT? Valid, but incorrect. If we apply it, we would not be able to construct the arc $John_1$ <- ate_2, because both 'John' and 'ate' would be on the stack, and in arc-eager, arcs are never created between two words on the stack

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

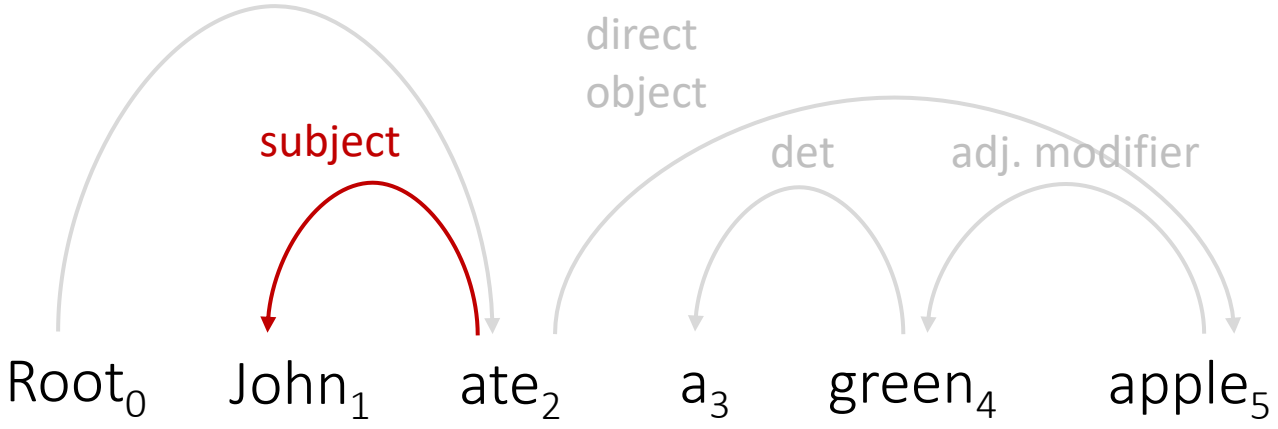
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}

LEFT-ARC? Valid and correct (there is an arc between $John_1$ <- ate_2)

RIGHT-ARC? Valid, but not correct (there is no arc $John_1$ -> ate_2)

REDUCE? Not valid: $John_1$ has not been assigned a parent yet.

SHIFT? Valid, but incorrect. If we apply it, we would not be able to construct the arc $John_1$ <- ate_2, because both 'John' and 'ate' would be on the stack, and in arc-eager, arcs are never created between two words on the stack

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

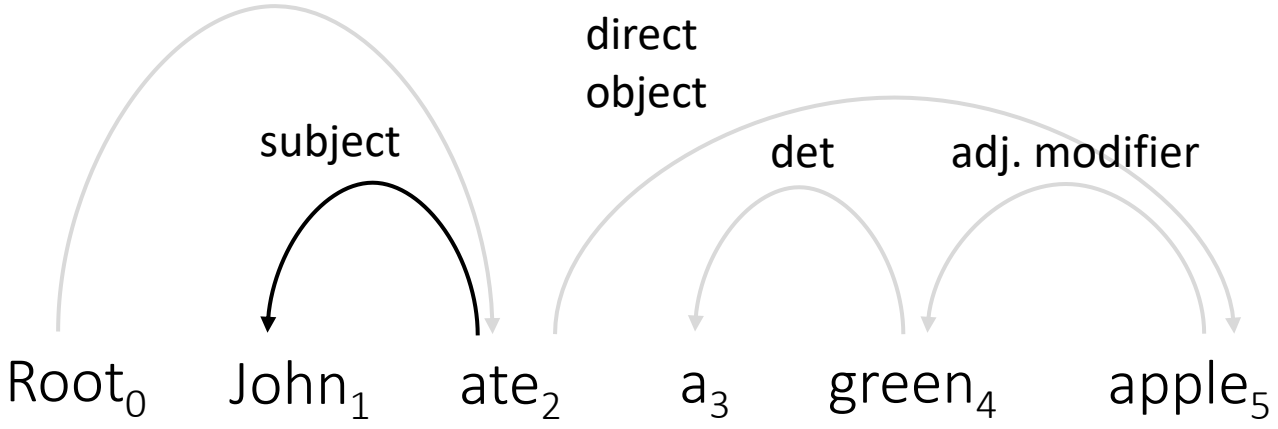
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]		

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

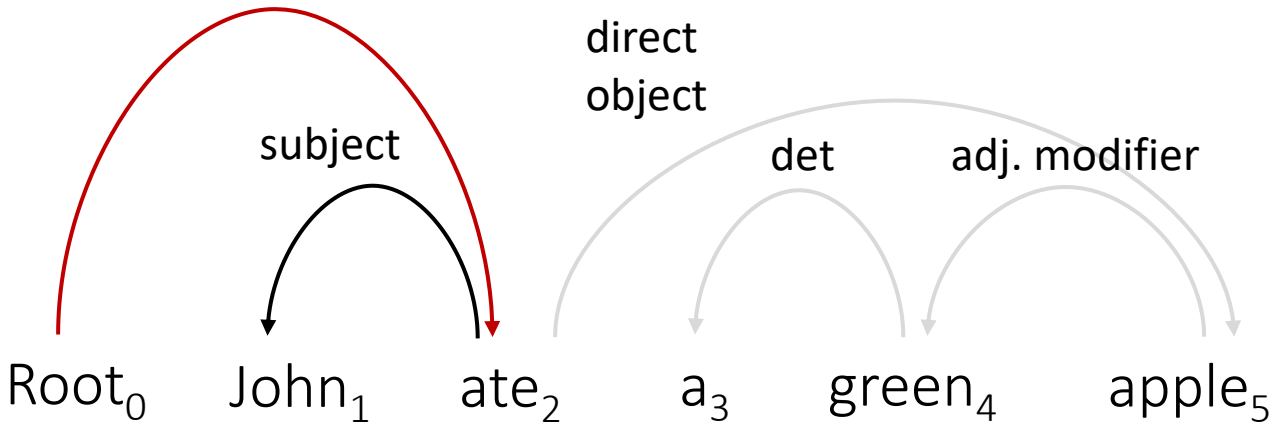
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

REDUCE = ∃k∃l[(k, l, i) ∈ A];

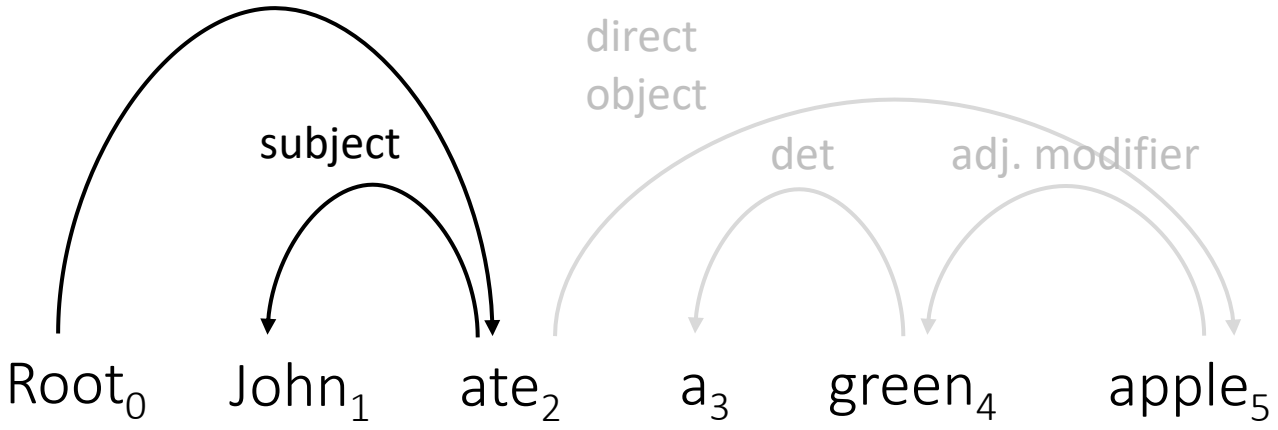
$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$

	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]		



Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

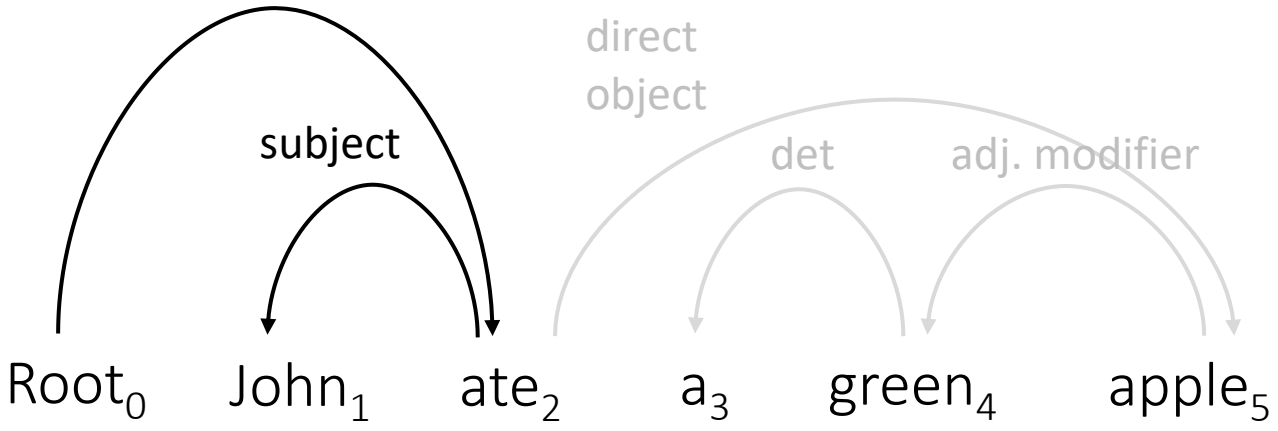
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

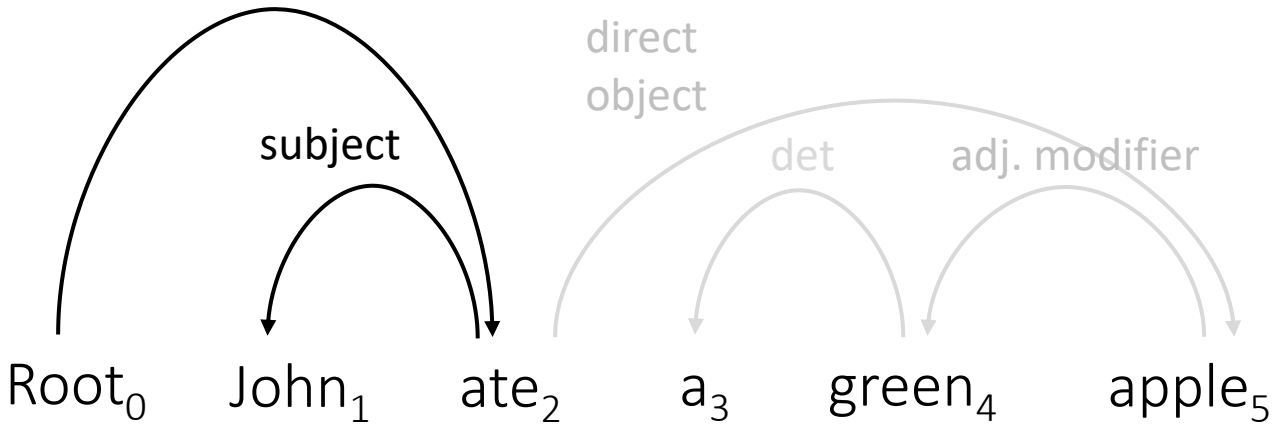
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]		

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

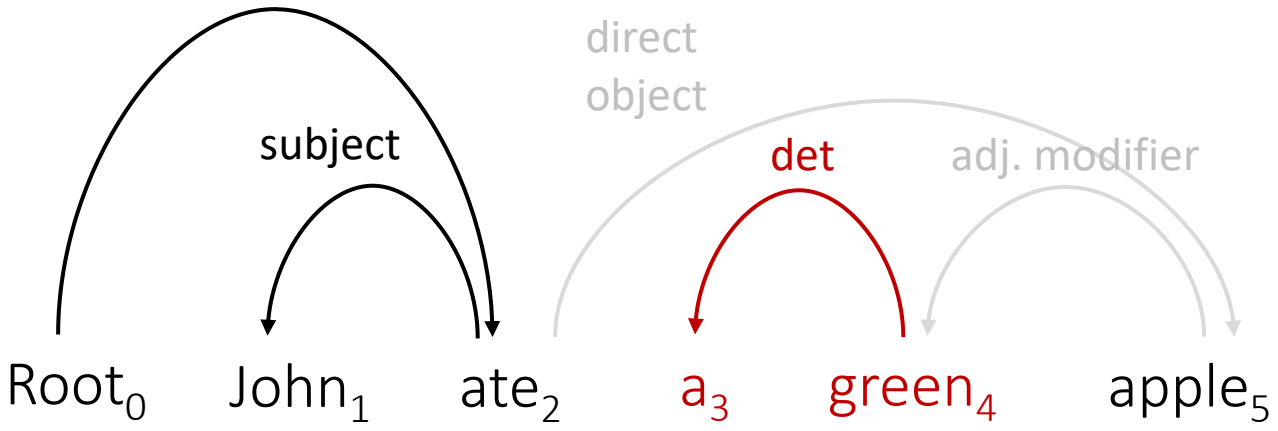
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

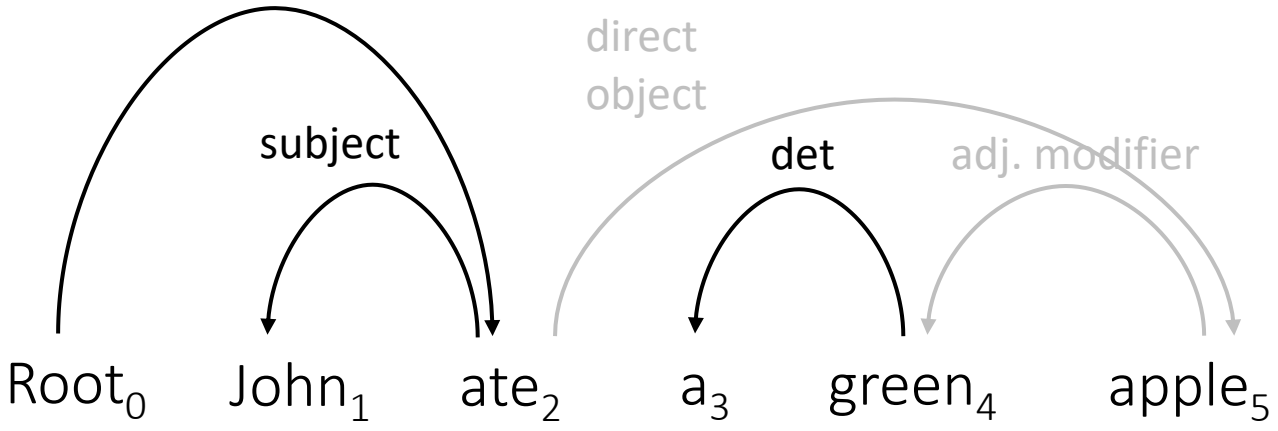
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]		

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

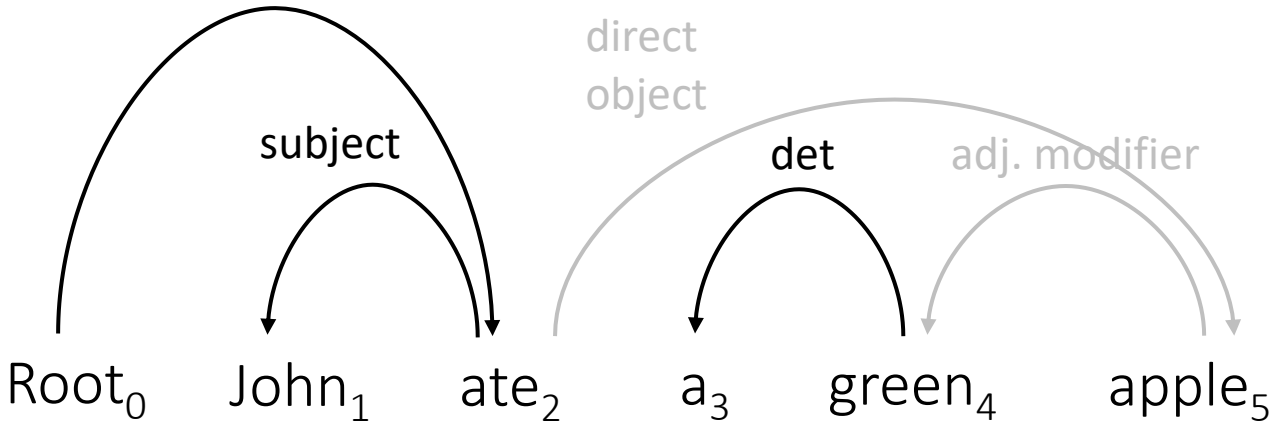
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

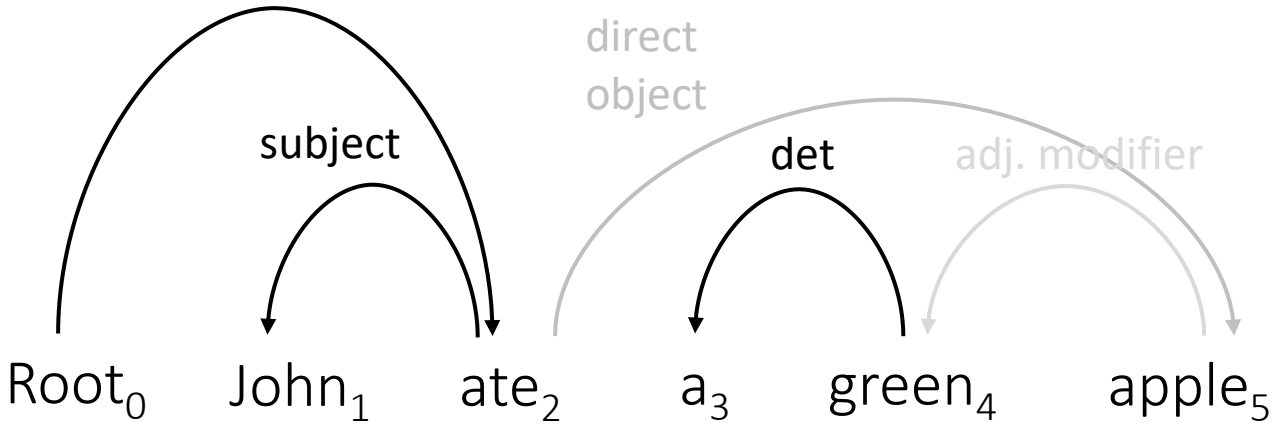
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}
7	[0,2,4]	[5]		

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

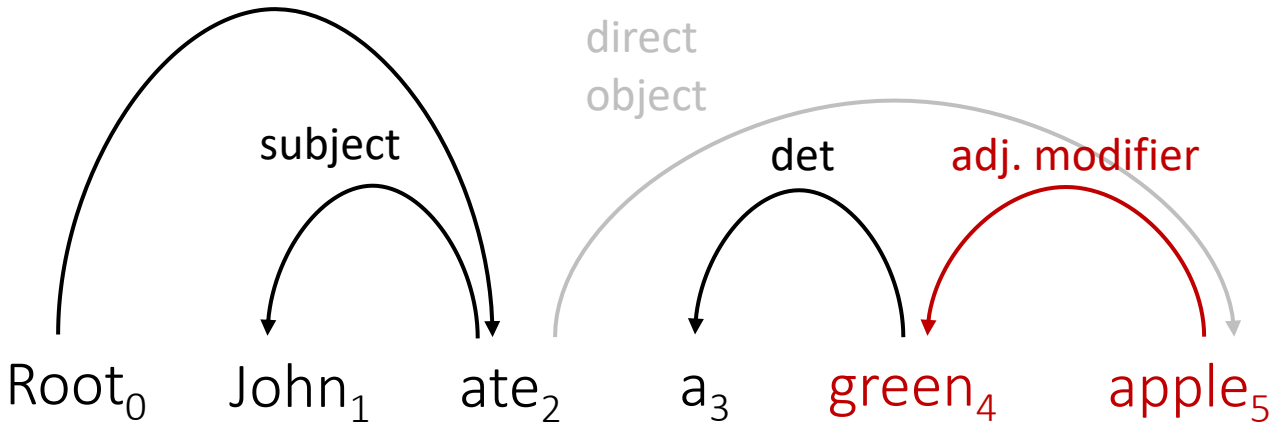
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}
7	[0,2,4]	[5]	LEFT-ARC	{(2,1), (0,2), (4,3), (5,4)}

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

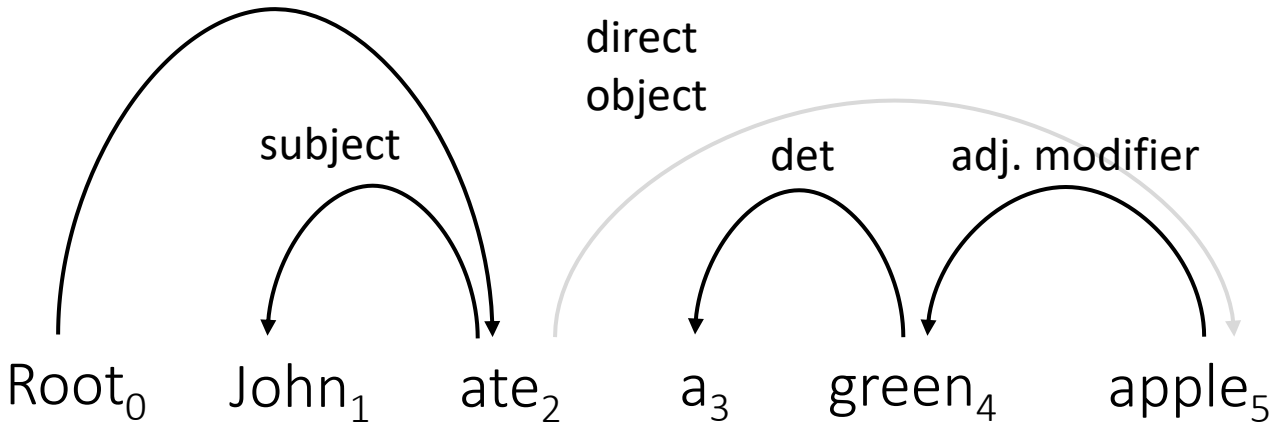
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}
7	[0,2,4]	[5]	LEFT-ARC	{(2,1), (0,2), (4,3), (5,4)}
8	[0,2]	[5]		

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

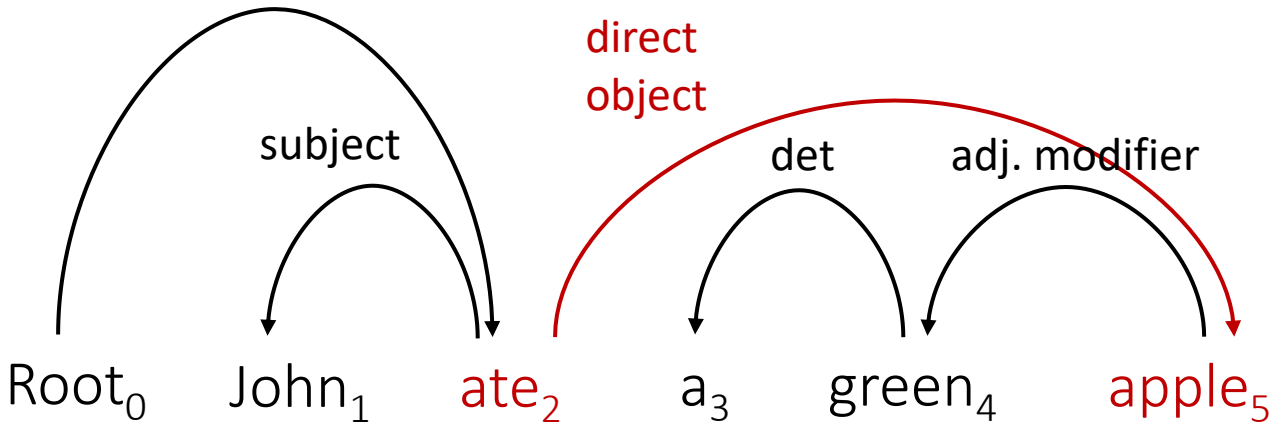
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}
7	[0,2,4]	[5]	LEFT-ARC	{(2,1), (0,2), (4,3), (5,4)}
8	[0,2]	[5]	RIGHT-ARC	{(2,1), (0,2), (4,3), (5,4), (2,5)}

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

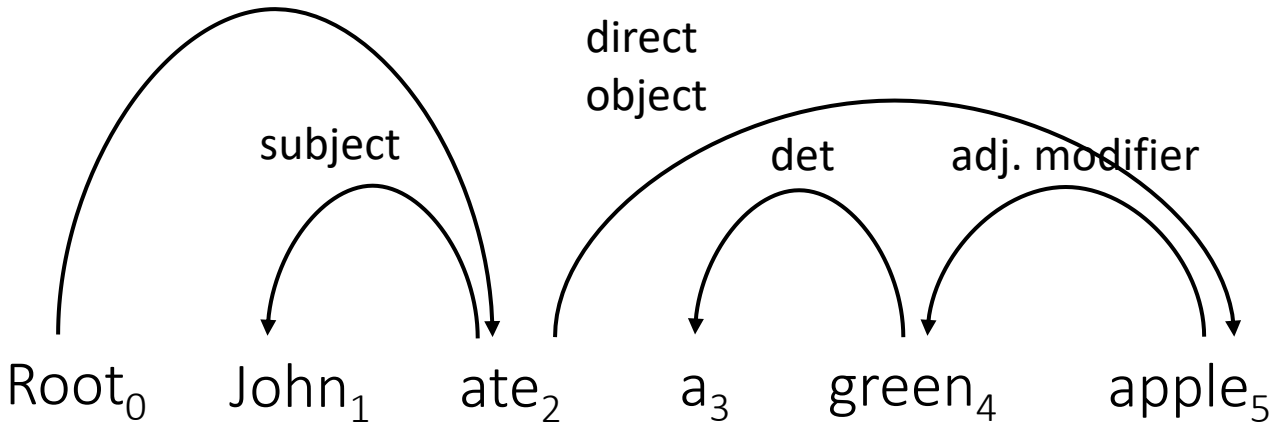
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}
7	[0,2,4]	[5]	LEFT-ARC	{(2,1), (0,2), (4,3), (5,4)}
8	[0,2]	[5]	RIGHT-ARC	{(2,1), (0,2), (4,3), (5,4), (2,5)}
9	[0,2,5]	[]		

Training - Oracle example

Preconditions

LEFT-ARC = ¬[i = 0];
¬∃k∃l [(k, l , i) ∈ A]

RIGHT-ARC = ¬∃k∃l [(k, l , j) ∈ A]

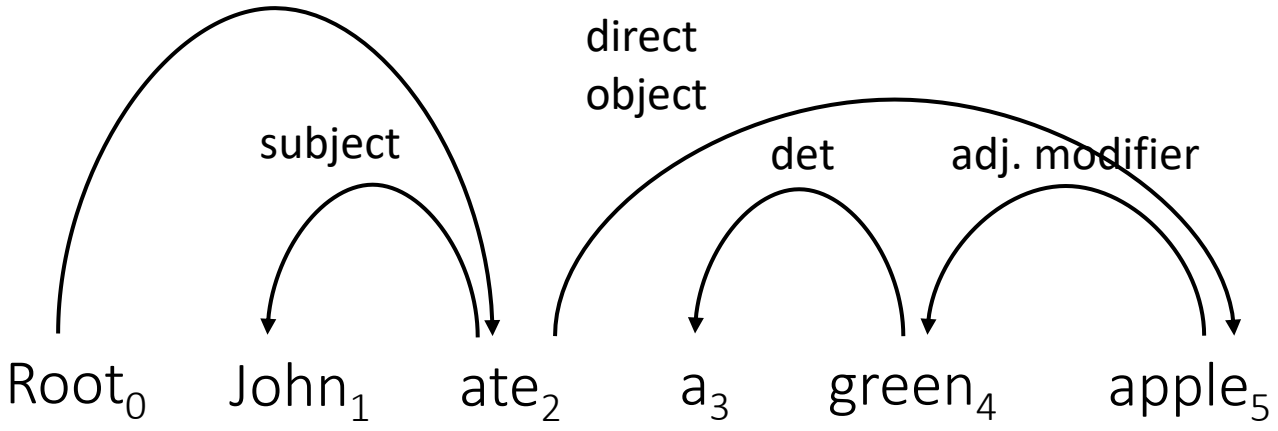
REDUCE = ∃k∃l[(k, l, i) ∈ A];

$$\text{LEFT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma, j|\beta, A \cup \{(j, l, i)\}]}$$

$$\text{RIGHT-ARC} = \frac{[\sigma|i, j|\beta, A]}{[\sigma|i|j, \beta, A \cup \{(i, l, j)\}]}$$

$$\text{REDUCE} = \frac{[\sigma|i, \beta, A]}{[\sigma, \beta, A]}$$

$$\text{SHIFT} = \frac{[\sigma, i|\beta, A]}{[\sigma|i, \beta, A]}$$



	Stack	Buffer	Action	Arcs
1	[0]	[1,2,3,4,5]	SHIFT	{}
2	[0,1]	[2,3,4,5]	LEFT-ARC	{(2,1)}
3	[0]	[2,3,4,5]	RIGHT-ARC	{(2,1),(0,2)}
4	[0,2]	[3,4,5]	SHIFT	{(2,1),(0,2)}
5	[0,2,3]	[4,5]	LEFT-ARC	{(2,1), (0,2), (4,3)}
6	[0,2]	[4,5]	SHIFT	{(2,1), (0,2), (4,3)}
7	[0,2,4]	[5]	LEFT-ARC	{(2,1), (0,2), (4,3), (5,4)}
8	[0,2]	[5]	RIGHT-ARC	{(2,1), (0,2), (4,3), (5,4), (2,5)}
9	[0,2,5]	[]	END	{(2,1), (0,2), (4,3), (5,4), (2,5)} *

*Notice the arcs in the Arcs set must include the label l, this is:
{(2,subject, 1), (0, root, 2), (4, det, 3), (5, adj.modifier, 4), (2, direct object, 5)}

Training - arc-eager algorithm

We have included a class ArcEager with an implementation template (algorithm.py)

```
class ArcEager():

    """
    Implements the arc-eager transition-based parsing algorithm for dependency parsing.

    This class includes methods for creating initial parsing states, applying transitions to
    these states, and determining the correct sequence of transitions for a given sentence.

    Methods:
        create_initial_state(sent: list[Token]): Creates the initial state for a given sentence.
        final_state(state: State): Checks if the current parsing state is a valid final configuration.
        LA_is_valid(state: State): Determines if a LEFT-ARC transition is valid for the current state.
        LA_is_correct(state: State): Determines if a LEFT-ARC transition is correct for the current state.
        RA_is_correct(state: State): Determines if a RIGHT-ARC transition is correct for the current state.
        RA_is_valid(state: State): Checks if a RIGHT-ARC transition is valid for the current state.
        REDUCE_is_correct(state: State): Determines if a REDUCE transition is correct for the current state.
        REDUCE_is_valid(state: State): Determines if a REDUCE transition is valid for the current state.
        oracle(sent: list[Token]): Computes the gold transitions for a given sentence.
        apply_transition(state: State, transition: Transition): Applies a given transition to the current state.
        gold_arcs(sent: list[Token]): Extracts gold-standard dependency arcs from a sentence.
    """
```

Training - Oracle template

```
state = self.create_initial_state(sent)

samples = [] #Store here all training samples for sent

#Applies the transition system until a final configuration state is reached
while not self.final_state(state):

    if self.LA_is_valid(state) and self.LA_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the LA transition using the function apply_transition
        raise NotImplementedError

    elif self.RA_is_valid(state) and self.RA_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the RA transition using the function apply_transition
        raise NotImplementedError

    elif self.REDUCE_is_valid(state) and self.REDUCE_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the REDUCE transition using the function apply_transition
        raise NotImplementedError
    else:
        #If no other transiton can be applied, it's a SHIFT transition
        transition = Transition(self.SHIFT)
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        samples.append(Sample(state, transition))
        #Update the state by applying the SHIFT transition using the function apply_transition
        self.apply_transition(state,transition)


#When the oracle ends, the generated arcs must
#match exactly the gold arcs, otherwise the obtained sequence of transitions is not correct
assert self.gold_arcs(sent) == state.A, f"Gold arcs {self.gold_arcs(sent)} and generated arcs {state.A} do not match"

return samples
```

SHIFT is already implemented

Training - Oracle template

```
state = self.create_initial_state(sent)

samples = [] #Store here all training samples for sent

#Applies the transition system until a final configuration state is reached
while not self.final_state(state):

    if self.LA_is_valid(state) and self.LA_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the LA transition using the function apply_transition
        raise NotImplementedError

    elif self.RA_is_valid(state) and self.RA_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the RA transition using the function apply_transition
        raise NotImplementedError

    elif self.REDUCE_is_valid(state) and self.REDUCE_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the REDUCE transition using the function apply_transition
        raise NotImplementedError
    else:
        #If no other transiton can be applied, it's a SHIFT transition
        transition = Transition(self.SHIFT)
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        samples.append(Sample(state, transition))
        #Update the state by applying the SHIFT transition using the function apply_transition
        self.apply_transition(state,transition)


#When the oracle ends, the generated arcs must
#match exactly the gold arcs, otherwise the obtained sequence of transitions is not correct
assert self.gold_arcs(sent) == state.A, f"Gold arcs {self.gold_arcs(sent)} and generated arcs {state.A} do not match"

return samples
```

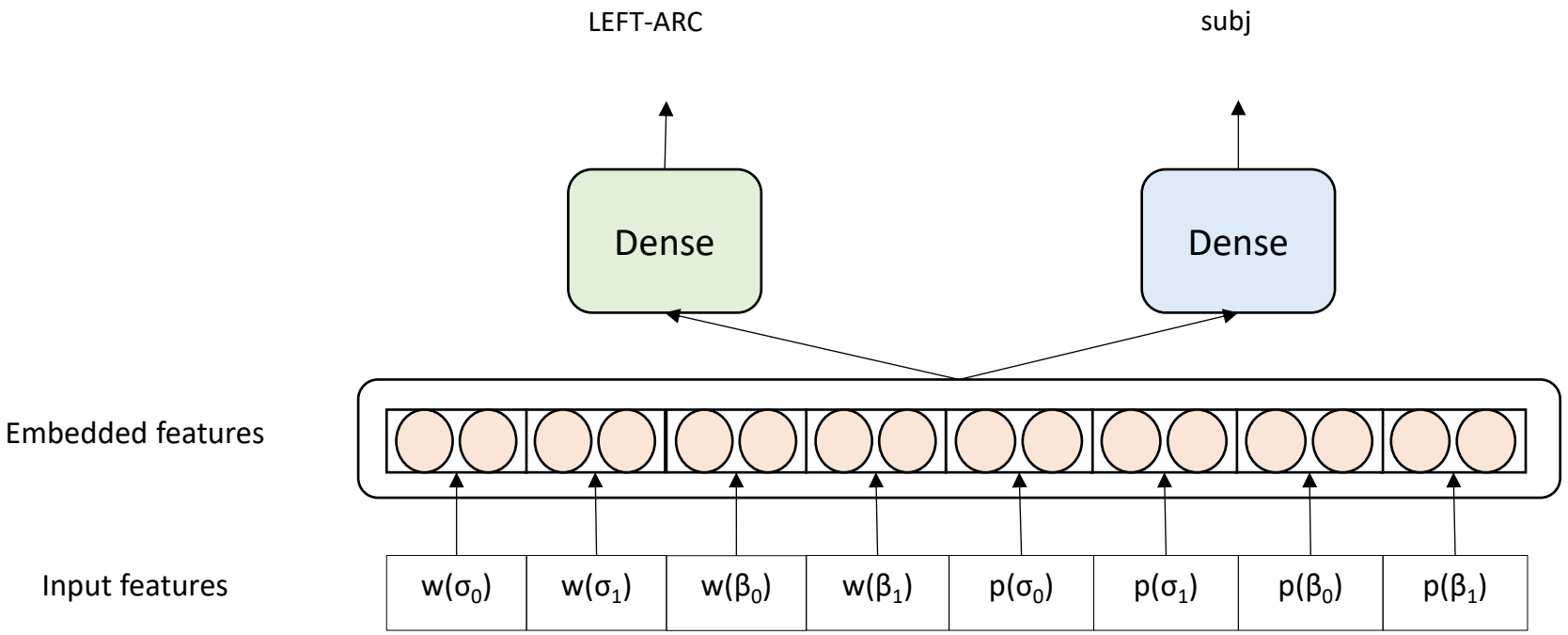
You have to implement this part

Training - Oracle template

```
state = self.create_initial_state(sent)

samples = [] #Store here all training samples for sent

#Applies the transition system until a final configuration state is reached
while not self.final_state(state):

    if self.LA_is_valid(state) and self.LA_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the LA transition using the function apply_transition
        raise NotImplementedError

    elif self.RA_is_valid(state) and self.RA_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the RA transition using the function apply_transition
        raise NotImplementedError

    elif self.REDUCE_is_valid(state) and self.REDUCE_is_correct(state):
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        #Update the state by applying the REDUCE transition using the function apply_transition
        raise NotImplementedError
    else:
        #If no other transiton can be applied, it's a SHIFT transition
        transition = Transition(self.SHIFT)
        #Add current state 'state' (the input) and the transition taken (the desired output) to the list of samples
        samples.append(Sample(state, transition))
        #Update the state by applying the SHIFT transition using the function apply_transition
        self.apply_transition(state,transition)


#When the oracle ends, the generated arcs must
#match exactly the gold arcs, otherwise the obtained sequence of transitions is not correct
assert self.gold_arcs(sent) == state.A, f"Gold arcs {self.gold_arcs(sent)} and generated arcs {state.A} do not match"

return samples
```

This assert checks if the Oracle-generated arcs are exactly as they should be.

Training - Proposed architecture

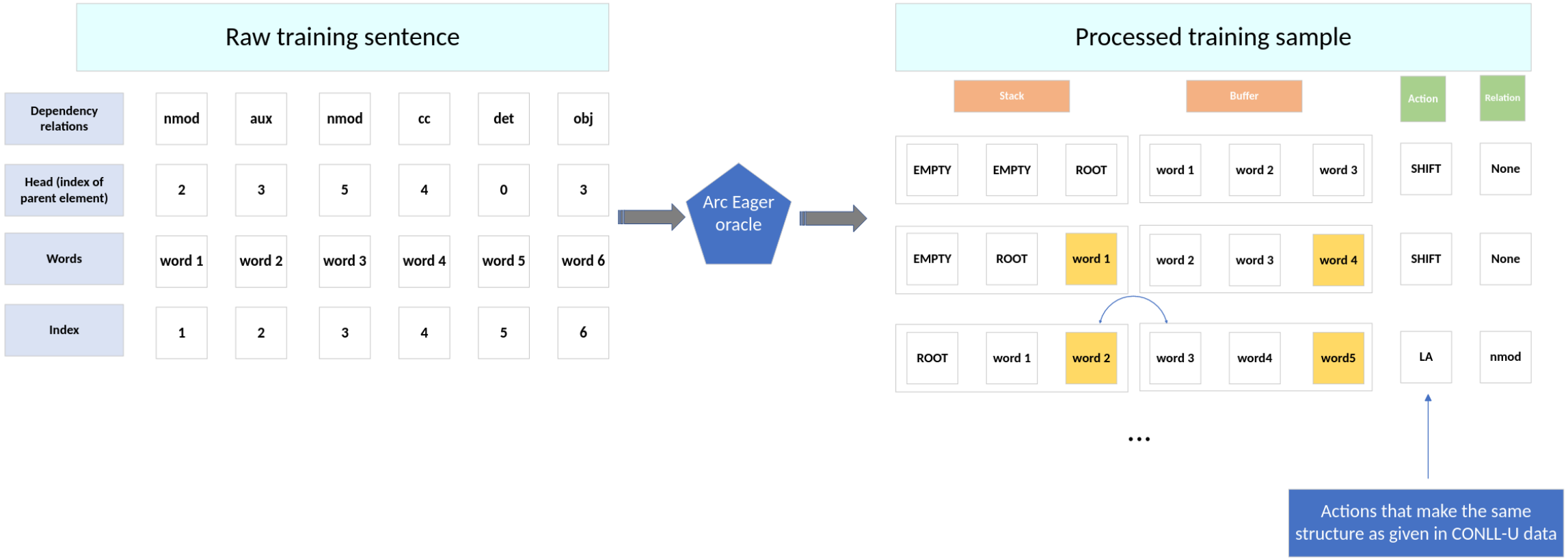


Key remarks:
- W is a function that retrieves the word form from a token and P is a function that retrieves the part-of-speech that from a token
- The number of features (from both the stack and the buffer) can be a design decision that you can establish
- Remember that Keras needs numbers, so make sure that the words, tags, transitions and dependency relationship labels are converted to an appropriate coding

Training - Preparing samples for our architecture

Raw training sentence						
Dependency relations	**nmod**	**aux**	**nmod**	**cc**	**det**	**obj**
Head (index of parent element)	**2**	**3**	**5**	**4**	**0**	**3**
Words	**word 1**	**word 2**	**word 3**	**word 4**	**word 5**	**word 6**
Index	**1**	**2**	**3**	**4**	**5**	**6**

Training - Preparing samples for our architecture



Training - Preparing samples for our architecture

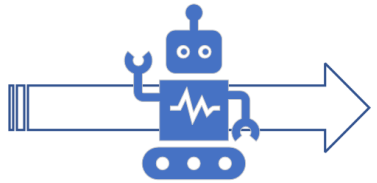
We have provided a partial implementation of the Sample class (algorithm.py), with training samples coded as (state, transition) pairs.

```
class Sample(object):
    """
    Represents a training sample for a transition-based dependency parser.

    This class encapsulates a parser state and the corresponding transition action
    to be taken in that state. It is used for training models that predict parser actions
    based on the current state of the parsing process.

    Attributes:
        state (State): An instance of the State class, representing the current parsing
                       state at a given timestep in the parsing process.
        transition (Transition): An instance of the Transition class, representing the
                                 parser action to be taken in the given state.

    Methods:
        state_to_feats(nbuffer_feats: int = 2, nstack_feats: int = 2): Extracts features from the parsing state.
    """
```

Training - Now it looks more familiar!

Movie classification:
Target variable #1: good/bad

word 1 | word 2 | word 3 | word 4 | word 5 | word 6

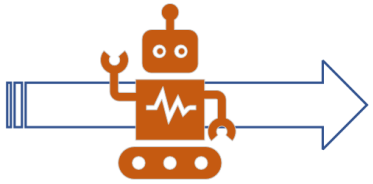


good
bad

Parsing action and dependency classification:
Target variable #1: LA, RA, Shift, Reduce
Target variable #2: None, nmod, cc ...

Neural Networks magic ....

ROOT | word 1 | word 2 | word 3 | word4 | word5



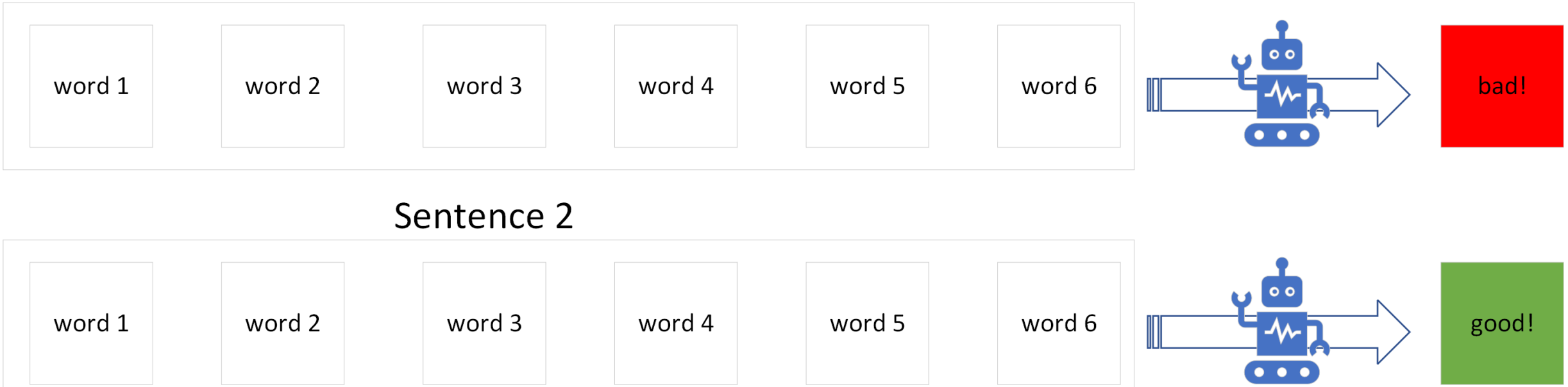
LA/RA/ Shift/ Reduce

None/ nmode/ ...

Training - But with a slight difference …

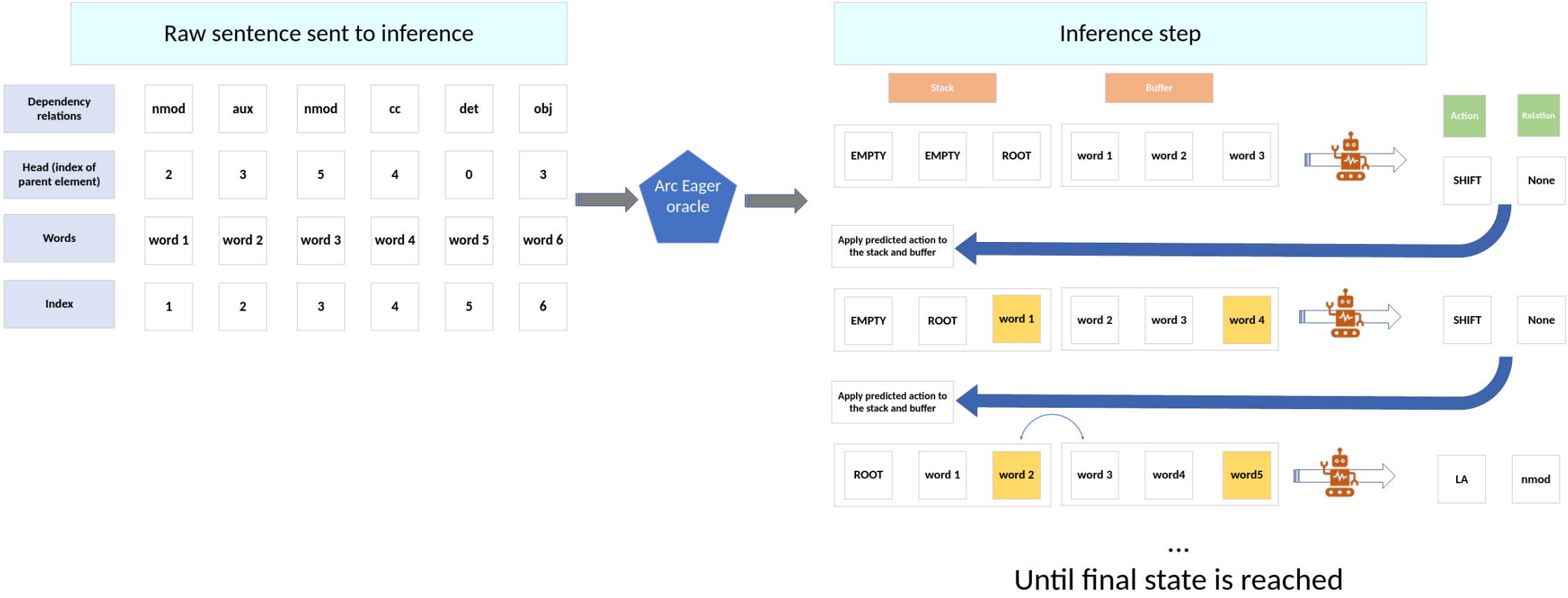
Classification needs one pass to make a prediction

Sentence 1



Sentence 2

Training - But with a slight difference …



…

Until final state is reached

Training - feature extraction

We need to extract features from samples, in order to provide the oracle's input.

- First-order features: directly extracted from the buffer and/or stack. E.g. the first two elements in the buffer and the two elements on top of the stack.

State: Stack (size=2): (0, ROOT, ROOT_UPOS) | (1, Distribution, NOUN)

Buffer (size=10): (4, license, NOUN) | (5, does, AUX)|...| (13, ., PUNCT)

Arcs (size=2): [(4, 'det', 3), (4, 'case', 2)]

Output: ['ROOT', 'Distribution', 'license', 'does', 'ROOT_UPOS', 'NOUN', 'NOUN', 'AUX']

Training - feature extraction

You will have to implement the method state_to_feats() of class Sample (algorithm.py), as indicated in the examples.

```python
def state_to_feats(self, nbuffer_feats: int = 2, nstack_feats: int = 2):
    """
    Extracts features from a given parsing state for use in a transition-based dependency parser.

    This function generates a feature representation from the current state of the parser,
    which includes features from both the stack and the buffer. The number of features from
    the stack and the buffer can be specified.

    Parameters:
        nbuffer_feats (int): The number of features to extract from the buffer.
        nstack_feats (int): The number of features to extract from the stack.

    Returns:
        list[str]: A list of extracted features. The features include the words and their
                corresponding UPOS (Universal Part-of-Speech) tags from the specified number
                of elements in the stack and buffer. The format of the feature list is as follows:
                [Word_stack_n,...,Word_stack_0, Word_buffer_0,...,Word_buffer_m,
                    UPOS_stack_n,...,UPOS_stack_0, UPOS_buffer_0,...,UPOS_buffer_m]
                where 'n' is nstack_feats and 'm' is nbuffer_feats.

    Examples:
        Example 1:
            State: Stack (size=1): (0, ROOT, ROOT_UPOS)
                Buffer (size=13): (1, Distribution, NOUN) | ... | (13, ., PUNCT)
                Arcs (size=0): []

            Output: ['<PAD>', 'ROOT', 'Distribution', 'of', '<PAD>', 'ROOT_UPOS', 'NOUN', 'ADP']

        Example 2:
            State: Stack (size=2): (0, ROOT, ROOT_UPOS) | (1, Distribution, NOUN)
                Buffer (size=10): (4, license, NOUN) | ... | (13, ., PUNCT)
                Arcs (size=2): [(4, 'det', 3), (4, 'case', 2)]

            Output: ['ROOT', 'Distribution', 'license', 'does', 'ROOT_UPOS', 'NOUN', 'NOUN', 'AUX']
    """
    raise NotImplementedError
```

Prediction – “horizontally” [inefficient, try to avoid it]

1. Take an initial sentence.
2. Obtain its initial state.
3. Extract the features and obtain the input representation.
4. Make a prediction with the trained model.
5. Verify that the predicted transition meets the preconditions and is valid (otherwise, establish a strategy to select an alternative transition).
6. Update the state and go back to step 3 until reaching a final state.

*This process is repeated independently for each sentence.

Prediction – “vertically” [efficient]

1. Take a batch of input sentences.
2. Obtain the corresponding batch of initial states.
3. Extract the batch of features from the current batch of states.
4. Make predictions with the trained model at the batch level.
5. Verify that the predicted transitions meet the preconditions.
6. Update the states.
7. Remove those instances from the batch that have reached a final state.
8. Repeat from step 3 until the batch is empty.

Prediction – “vertically” [efficient] - example

1. Take a batch of input sentences:

batch_sentences = ["John ate a green apple", "Mary is sick"]

2. Obtain the corresponding batch of initial states:

batch_states= [(Stack=ROOT, Buffer = John|ate|a|green|apple, Arcs={}) ,
(Stack=ROOT, Buffer=Mary|is|sick), Arcs={}]

3. Extract the batch of features from the current batch of states:

batch_state_feats= [[<PAD>, ROOT, John, ate, <PAD>, ROOT_UPOS, NOUN, VERB],
[<PAD>, ROOT, Mary, is, <PAD>, ROOT_UPOS, NOUN, VERB]]

4. Make predictions with the trained model at the batch level (obtain the probability distribution for each possible transition and dependency type):

batch_actions, batch_deprels = model.predict(batch_state_feats)

Prediction – “vertically” [efficient] - example

5. Select the most probable dependency type and valid (following arc-eager preconditions) transition. E.g., assuming transitions ordered by probability:

probs_transitions= [[SHIFT (0.4), RA (0.3), LA (0.2), REDUCE (0.1)], [~~LA (0.3),~~ SHIFT(0.25), REDUCE (0.25), RA (0.2)]]

6. Update states with selected transitions:

batch_selected_transitions (SHIFT, SHIFT)

batch_states = [(Stack=ROOT, Buffer = John|ate|a|green|apple, Arcs={}), (Stack=ROOT, Buffer=Mary|is|sick), Arcs={}]

new_states = [(Stack=ROOT|John, Buffer = ate|a||green|apple, Arcs={}), (Stack=ROOT|Mary Buffer=is|sick), Arcs={}]

batch_states = new_states

7. Remove those instances from the batch that have reached a final state (there are none in this example).

8. Repeat from step 3 until the batch is empty.

Output and post-processing

In order to evaluate the accuracy of our model on the test set, we must first write the output in a CoNLL-U format file.

To do this, we need to implement a function that transforms a sequence of transitions into a dependency tree. The function is similar to the oracle function.

We receive as input a sentence, along with its initial state and the predicted sequence of transitions:

1. We apply the next transition from the list and update the state. If an arc has been created, we save it.

2. We repeat the process until a final state is reached.

Output and post-processing

When making predictions, we may obtain corrupted trees. These will need to be corrected using heuristics in order to obtain well-formed trees.

Some examples of situations causing ill-formed trees:

- Nodes without an assigned parent (it is easy to imagine an extreme case, a sequence of transitions where we only perform SHIFT).
- Sentences in which no term is assigned ROOT as its parent.
- Sentences in which multiple terms are assigned ROOT as their parent. This is not strictly incorrect, but in Universal Dependencies (UD), trees are expected to be single-rooted. Therefore, you must control for this, or there will be issues in the evaluation.
- In the arc-eager algorithm, there is no possibility of forming cycles, but that is possible in other transition-based algorithms.

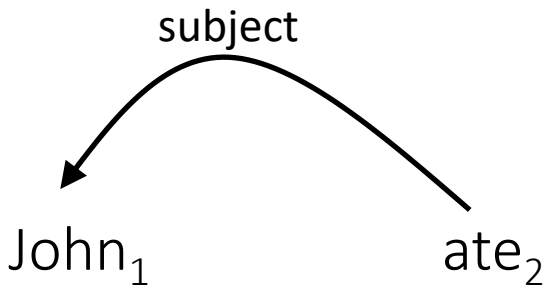
Postprocessing (already implemented in postprocessor.py)

The postprocessing module is also implemented within the Postprocessor class in postprocessor.py. An example of usage is included at the end of the file.

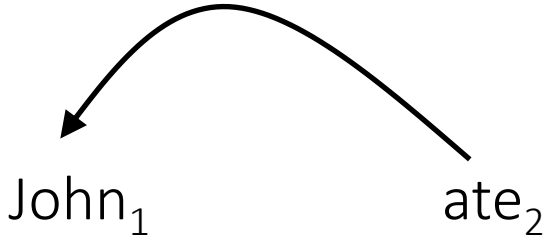
```
class PostProcessor():
    """
    A class for post-processing syntactic trees parsed from CoNLL-U formatted files.

    This class provides functionality to correct trees that have issues such as
    multiple roots or tokens without an assigned head. It ensures that each tree
    conforms to a standard structure with a single root and all tokens having
    an assigned head.

    Methods:
        postprocess: Corrects the trees in a given CoNLL-U file and returns the corrected trees.
    """
```

Evaluation

LAS (Labeled Attachment Score): The percentage of words for which both the HEAD and DEPREL (dependency relation label) are correctly predicted.



UAS (Unlabeled Attachment Score): The percentage of words for which the HEAD (parent) is correctly predicted, regardless of the specific dependency relation label (DEPREL).



For evaluation, you should use the official evaluation script conll18_ud_eval.py, which is available at: https://universaldependencies.org/conll18/evaluation.html.

Evaluation

$ python conll18_ud_eval.py ./data/en_partut-ud-test_clean.conllu .\data\en_partut-ud-test_clean.conllu -v

Metric	Precision	Recall	F1 Score	AlignAcc
Tokens	100.00	100.00	100.00	
Sentences	100.00	100.00	100.00	
Words	100.00	100.00	100.00	
UPOS	100.00	100.00	100.00	100.00
XPOS	100.00	100.00	100.00	100.00
UFeats	100.00	100.00	100.00	100.00
AllTags	100.00	100.00	100.00	100.00
Lemmas	100.00	100.00	100.00	100.00
UAS	100.00	100.00	100.00	100.00
LAS	100.00	100.00	100.00	100.00
CLAS	100.00	100.00	100.00	100.00
MLAS	100.00	100.00	100.00	100.00
BLEX	100.00	100.00	100.00	100.00

- Notice that the underlined parameter is the path to the file with your system's predictions.
  - Departing from the test data, simply update the head and deprel fields according to what your parser obtains, and save it
- Depending on how you design and train your neural model, you may expect scores ranging between 55-85

Schedule

Most difficult parts of the assignment.

Make sure they are completed on time.

Otherwise, it is very likely that you will have issues finishing this assignment before the deadline.

Weeks	Tasks that should be completed at this point
November, 5th	Presentation (5th) and familiarization with the assignment.
November, 12th	Implementation of the oracle, feature extraction, and obtaining training samples for a Keras model.
November, 19th	Implementation, training, and evaluation of the neural network.
November, 26th	Implementation of decoding, and evaluation with the conll18_ud_eval.py script.
December, 3rd	Report with results analysis and final questions.
December, 10th	**Submission (9th)** and defense (10th).

Bibliography

[1] Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553. https://aclanthology.org/J08-4003

[2] Jurafsky & Martin. Speech and Language Processing. Chapter 19: Dependency Parsing https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

[3] Kübler, McDonald & Nivre. Synthesis. Dependency Parsing. Lectures on Human Language Technologies. Springer, 2009.