UPPSALA UNIVERSITET

Transition-based dependency parsing

Syntactic analysis/parsing

2023

Sara Stymne
Department of Linguistics and Philology

Based on slides from Marco Kuhlmann

UPPSALA UNIVERSITET

Transition-based dependency parsing

UPPSALA UNIVERSITET

Transition-based dependency parsing

- Eisner's algorithm runs in time $O(|w|^3)$.
  This may be too much if a lot of data is involved.
- Idea: Design a dumber but really fast algorithm and let the machine learning do the rest.
- Eisner's algorithm searches over many different dependency trees at the same time.
- A transition-based dependency parser only builds *one* tree, in *one* left-to-right sweep over the input.

Transition-based dependency parsing

- The parser starts in an initial configuration.
- At each step, it asks a guide to choose between one of several transitions (actions) into new configurations.
- Parsing stops if the parser reaches a terminal configuration.
- The parser returns the dependency tree associated with the terminal configuration.

Generic parsing algorithm

```
Configuration c = parser.getInitialConfiguration(sentence)

while c is not a terminal configuration do

     Transition t = guide.getNextTransition(c)

     c = c.makeTransition(t)

return c.getGraph()
```

Variation

Transition-based dependency parsers differ with respect to the configurations and the transitions that they use.

UPPSALA
UNIVERSITET

The arc-standard algorithm

The arc-standard algorithm

- The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.
- It is very similar to shift–reduce parsing as it is known for context-free grammars.
- It is implemented in most practical transition-based dependency parsers, including MaltParser.

Configurations

A configuration for a sentence $w = w_1 \ldots w_n$ consists of three components:

- a buffer containing words of w
- a stack containing words of w
- the dependency graph constructed so far

UPPSALA UNIVERSITET

Configurations

- Initial configuration:
    - All words are in the buffer.
    - The stack is empty.
    - The dependency graph is empty.
- Terminal configuration:
    - The buffer is empty.
    - The stack contains a single word.

Possible transitions

- shift (sh): push
  the next word in the buffer onto the stack
- left-arc (la): add an arc
  from the topmost word on the stack, s_1,
  to the second-topmost word, s_2, and pop s_2
- right-arc (ra): add an arc
  from the second-topmost word on the stack, s_2,
  to the topmost word, s_1, and pop s_1

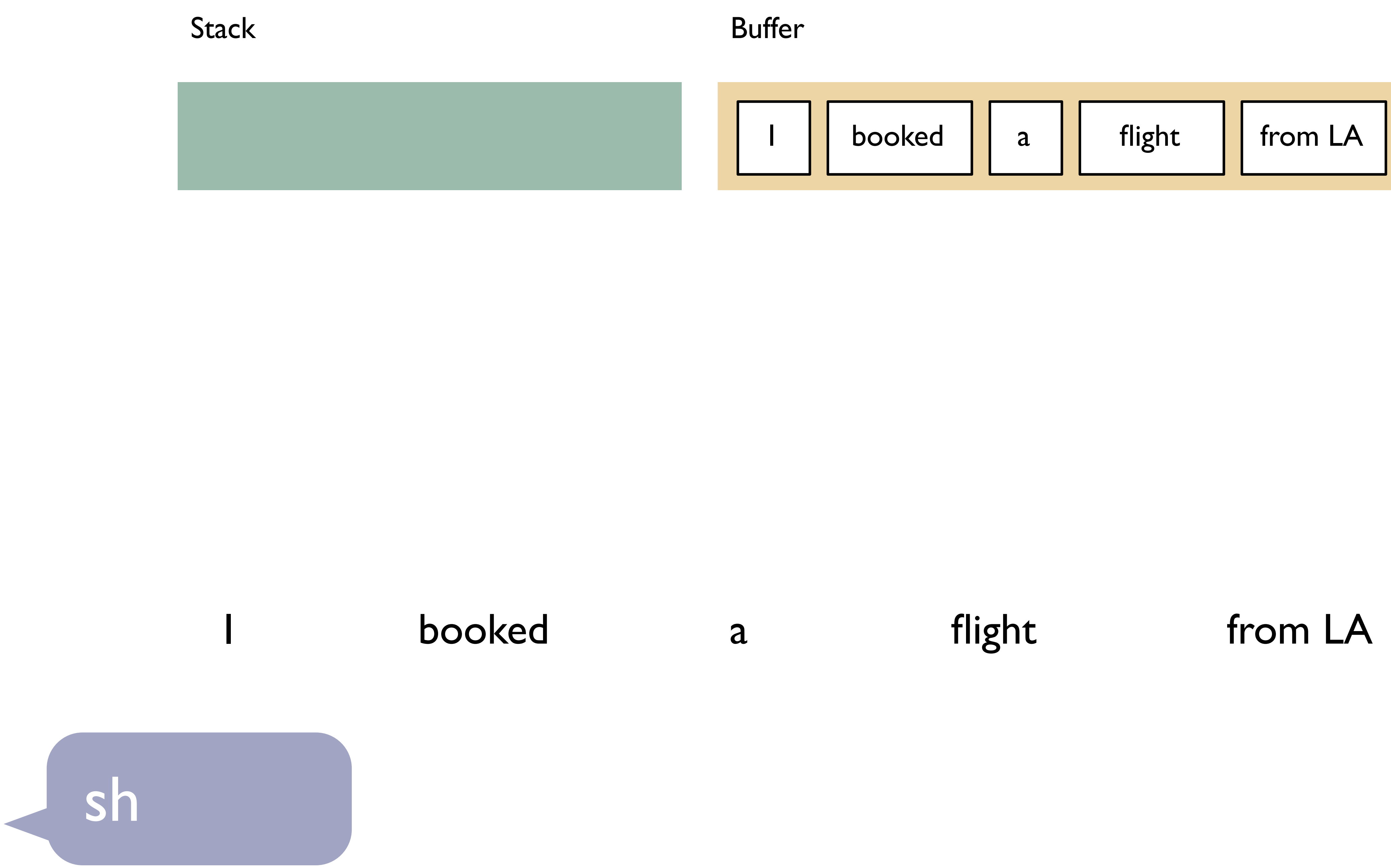
Terminology

- Stack
  - S - the full stack
  - σ - partial stack
  - [σ|i|j] - a generic stack σ, with elements i,j on top (opening to right)
- Buffer
  - B - full buffer
  - β - partial buffer
  - [i|β] - buffer with element i as the first element (opening to left)

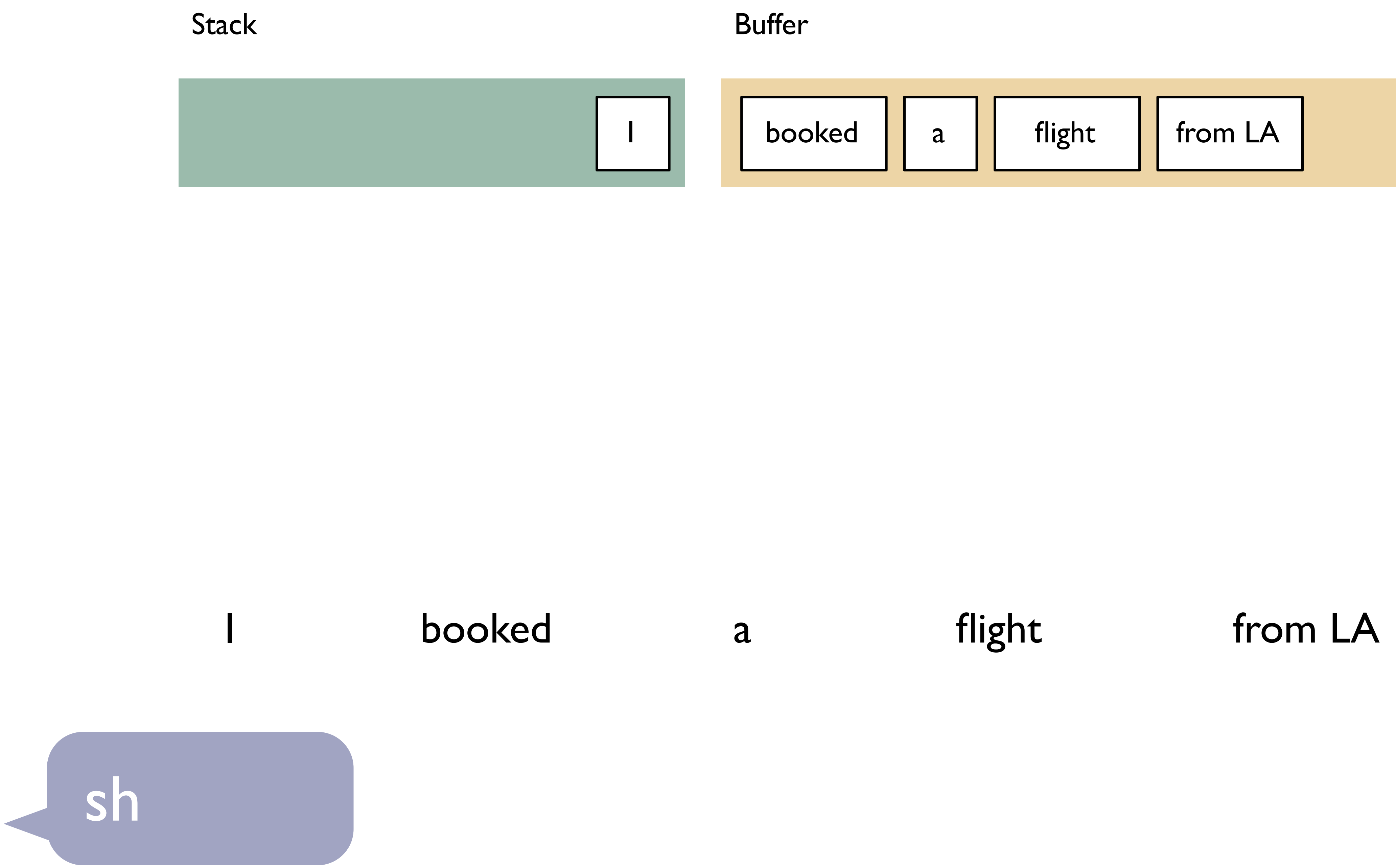
Configurations and transitions

- Initial configuration: ([],[0,...,n],[])
- Terminal configuration: ([i],[],A)
- shift (sh):
  (σ,[i|β],A) ⇒ ([σ|i],β,A)
- left-arc (la):
  ([σ|i|j],B,A) ⇒ ([σ|j],B,A∪{j,l,i})
- right-arc (ra):
  ([σ|i|j],B,A) ⇒ ([σ|i],B,A∪{i,l,j})

Example run


Stack
Buffer
I
booked
a
flight
from LA
I
booked
a
flight
from LA
sh

Example run


Stack
Buffer
I
booked
a
flight
from LA
I
booked
a
flight
from LA
sh

UPPSALA UNIVERSITET

The arc-standard algorithm

Example run

Stack

I | booked

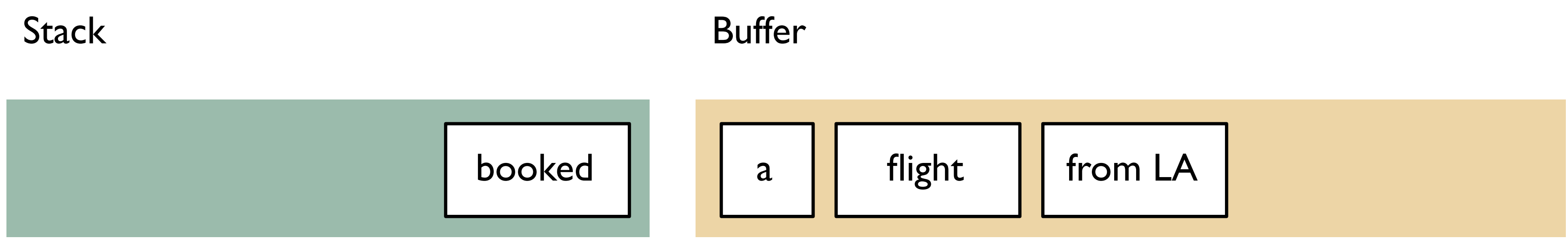
Buffer

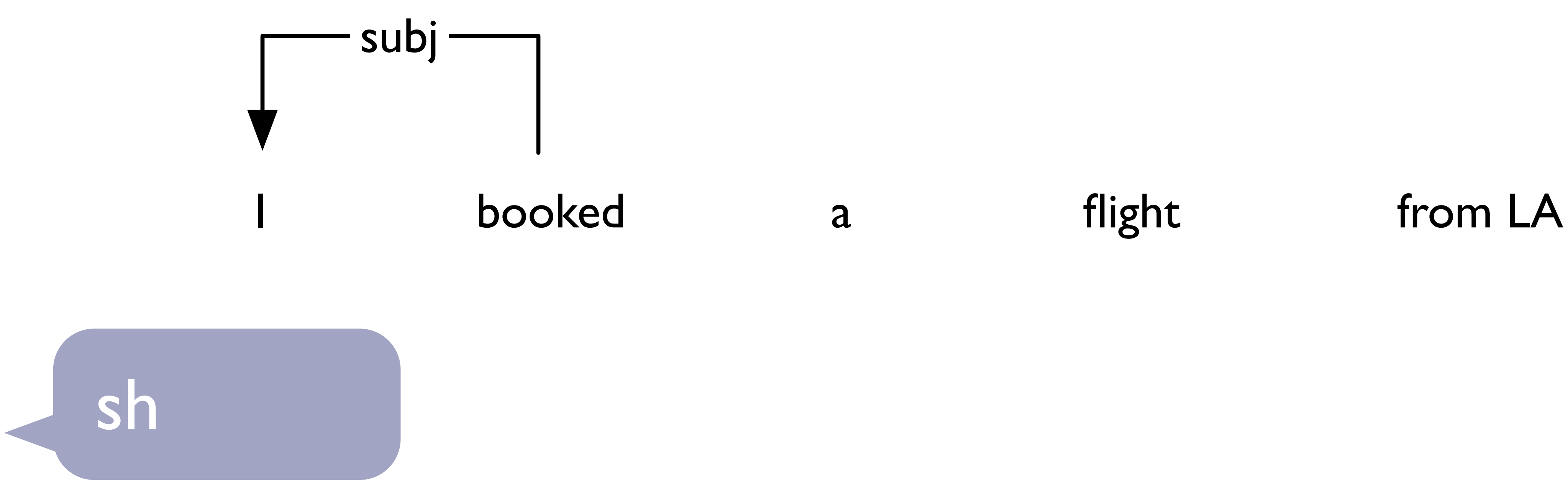
a | flight | from LA

I booked a flight from LA

la-subj

Example run

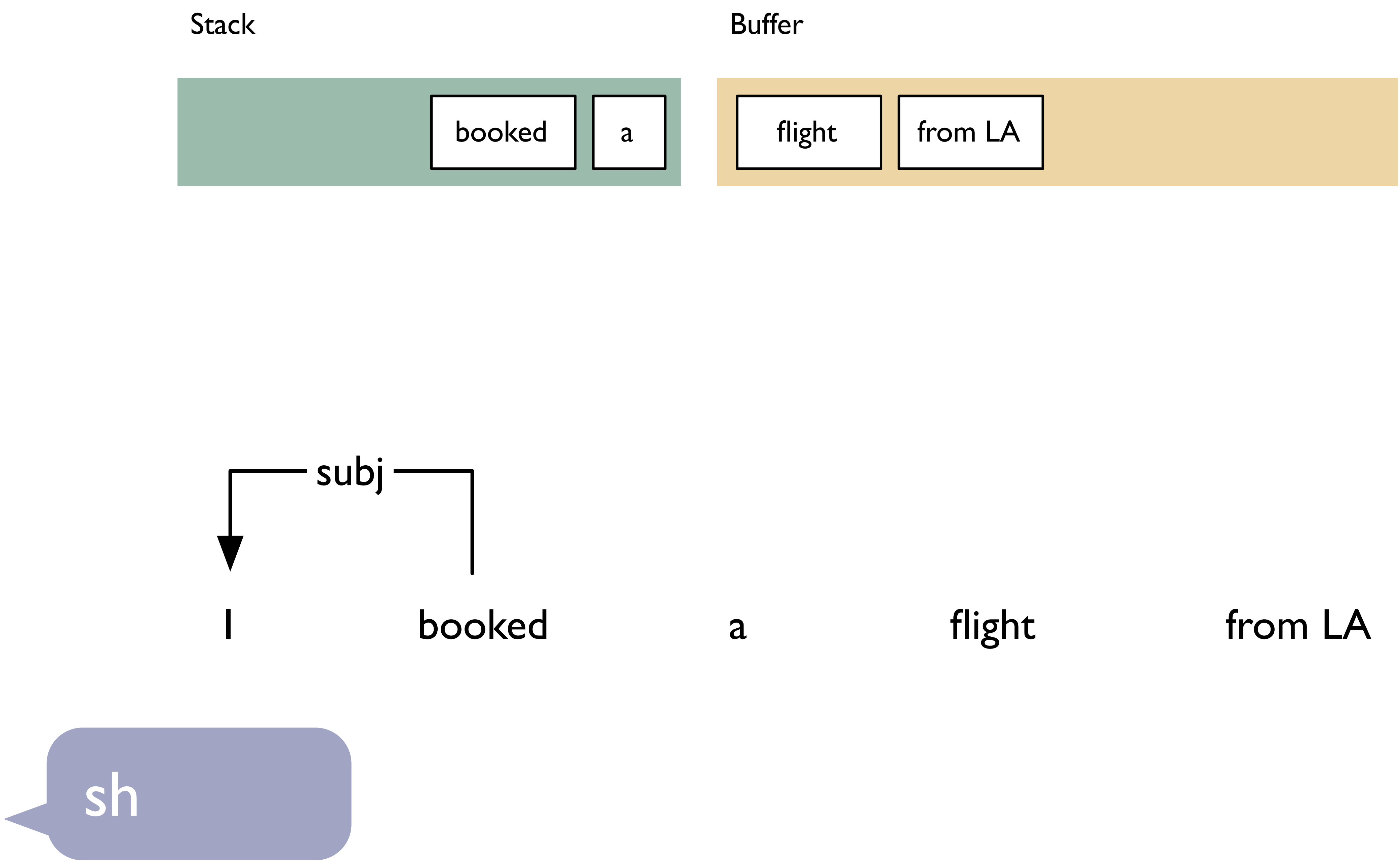

Stack
booked
Buffer
a
flight
from LA


subj
I
booked
a
flight
from LA
sh

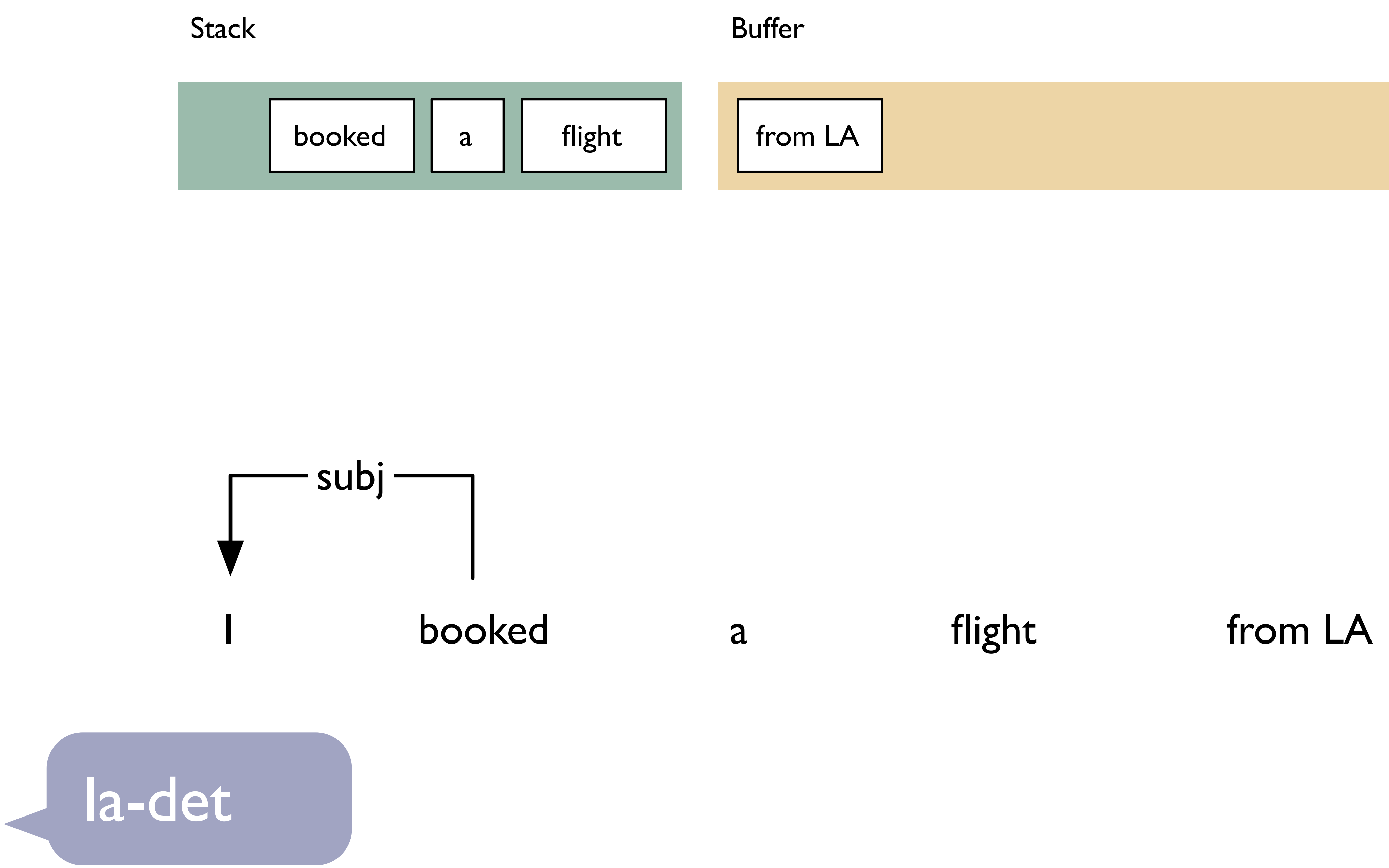
UPPSALA
UNIVERSITET

The arc-standard algorithm

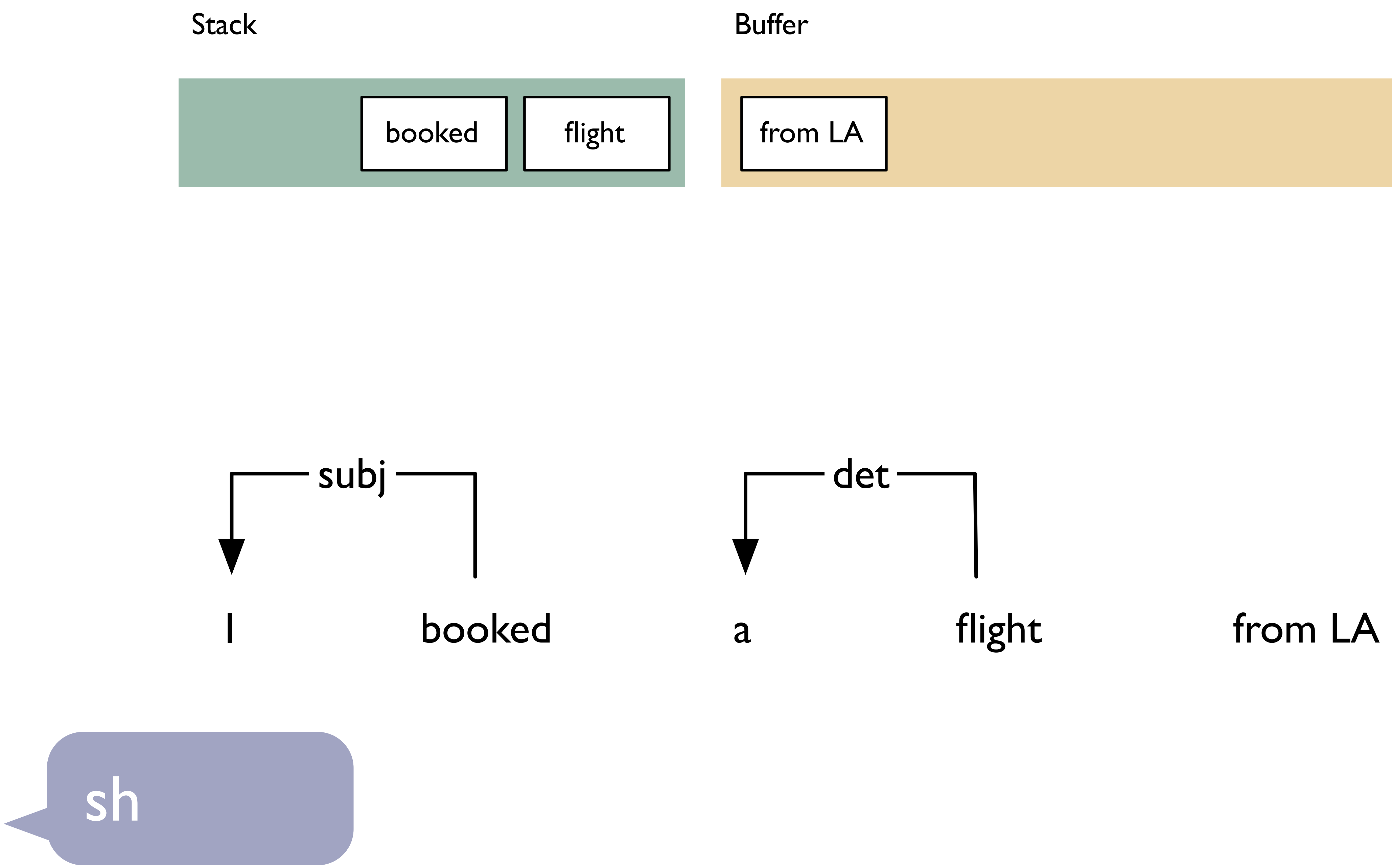
Example run



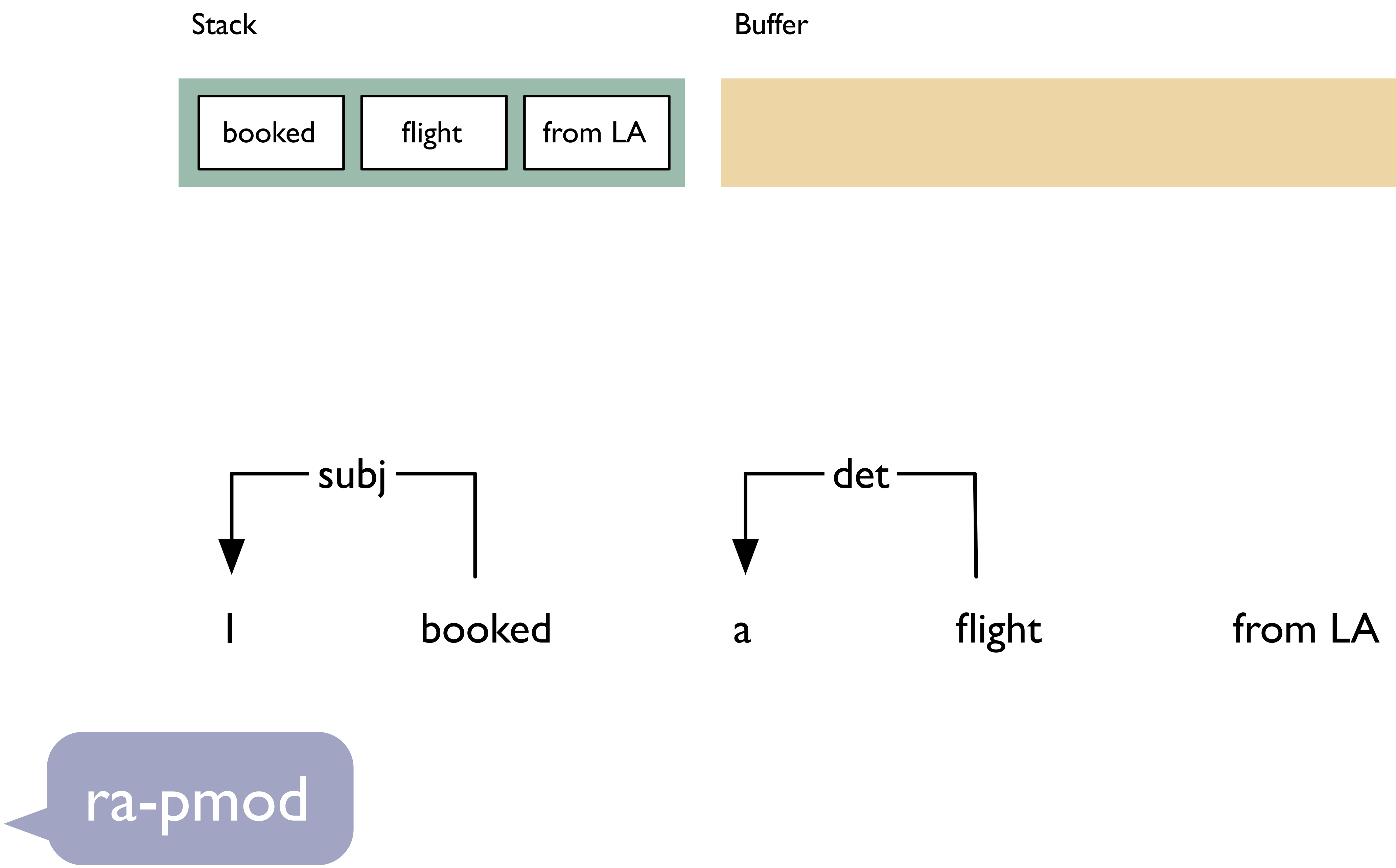
Example run


Stack
booked
a
flight
Buffer
from LA
subj
I
booked
a
flight
from LA
la-det

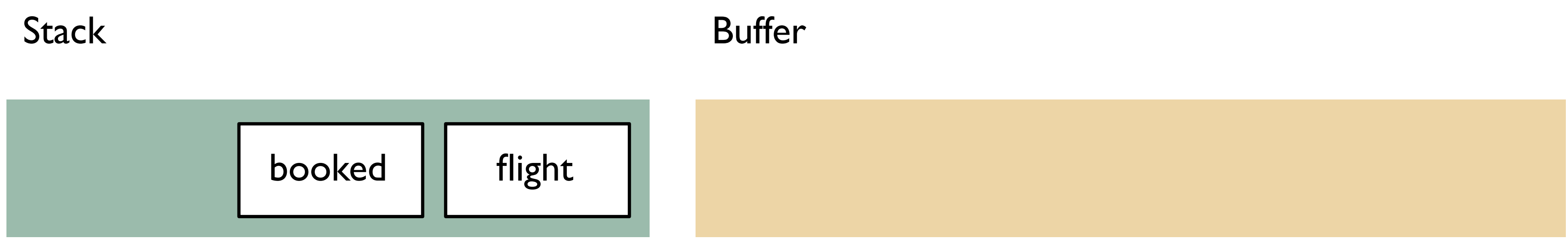
Example run

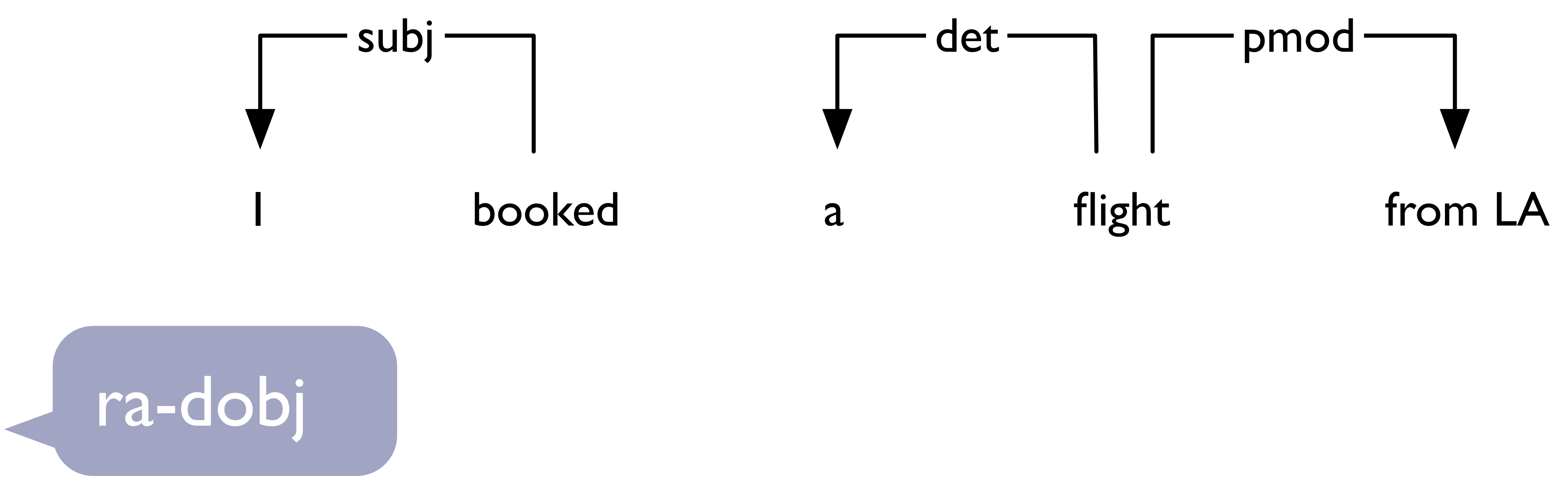

Stack
Buffer
booked
flight
from LA
subj
det
I
booked
a
flight
from LA
sh

Example run


Stack
Buffer
booked
flight
from LA
subj
det
I
booked
a
flight
from LA
ra-pmod

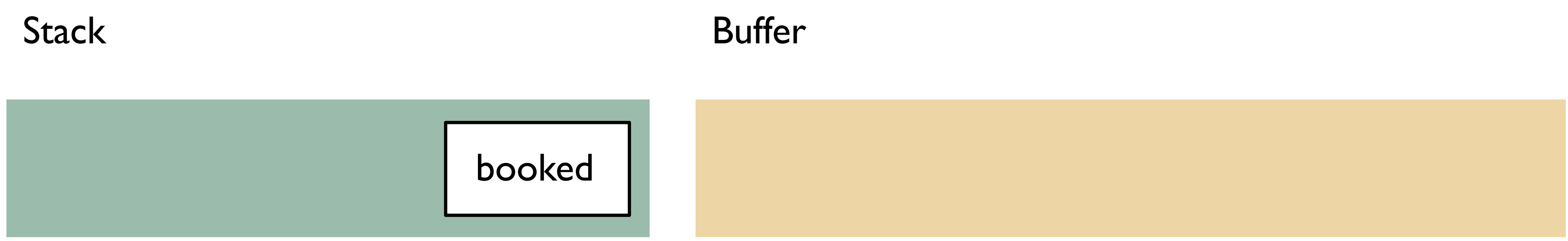
Example run

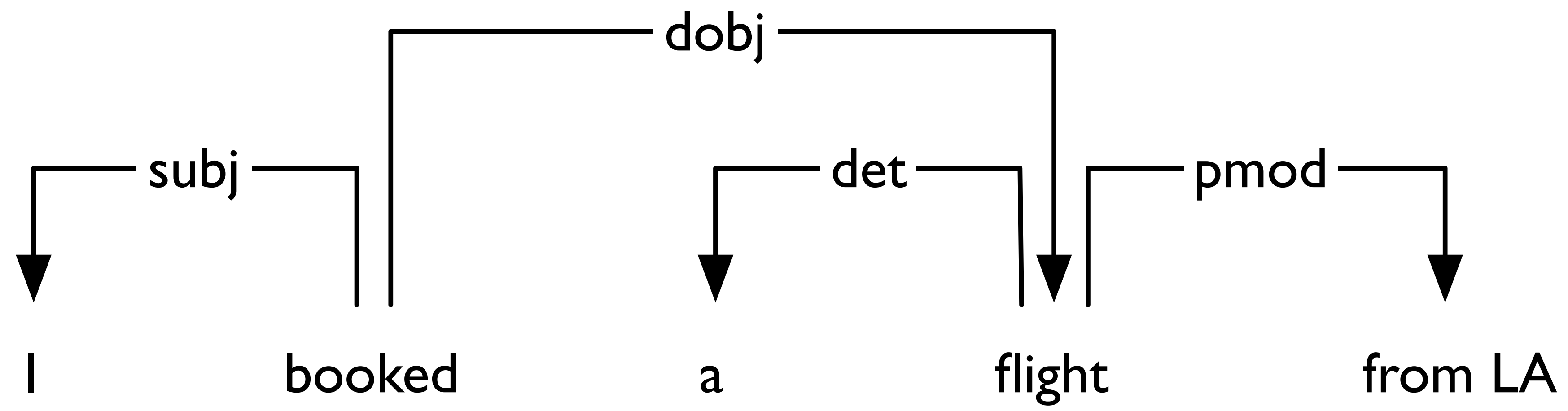

Stack
booked
flight
Buffer


subj
det
pmod
I
booked
a
flight
from LA
ra-dobj

The arc-standard algorithm

Example run





UPPSALA UNIVERSITET

Complexity and optimality

- Time complexity is linear, O(n), since we only have to treat each word once
- This can be achieved since the algorithm is greedy, and only builds one tree, in contrast to Eisner's algorithm, where all trees are explored
- There is no guarantee that we will even find the best tree given the model, the arc-standard model.
- There is a risk of error propagation
- An advantage is that we can use very informative features, for the ML algorithm

UPPSALA UNIVERSITET

Training a guide

Guides

- We need a guide that tells us what the next transition should be.
- The task of the guide can be understood as classification: Predict the next transition (class), given the current configuration.

Training a guide

- We let the parser run on gold-standard trees.
- Every time there is a choice to make, we simply look into the tree and do ‘the right thing’™. (oracle)
- We collect all (configuration, transition) pairs and train a classifier on them.
- When parsing unseen sentences,
  we use the trained classifier as a guide.

Example features

	Attributes				
Adress	**FORM**	**LEMMA**	**POS**	**FEATS**	**DEPREL**
Stack[0]	X	X	X	X	
Stack[1]	X		X		
Ldep(Stack[0])					X
Rdep(Stack[0])					X
Buffer[0]	X	X	X	X	
Buffer[1]			X		
. . .					

- Combinations of addresses and attributes (e.g. those marked in the table)
- Other features, such as distances, number of children, ...

UPPSALA UNIVERSITET

Training with neural networks

- Neural networks are a good fit for the classification tasks in transition-based features
- Features can, for instance, be extracted for each word from recurrent neural networks (RNN), or transformers
    - Each word is partially represented by its context

UPPSALA
UNIVERSITET

Alternative transition models and oracles

Transition models in Maltparser

- Arcs between two topmost words on stack
  - arc-standard model (variant from slides)
  - models with swap transition
- Arcs created between stack and buffer
  - arc-eager model
  - arc-standard (variant from course book)
- Mix
  - arc-hybrid
  - arc-hybrid + swap

UPPSALA UNIVERSITET

Arc-eager model

- Contains four transitions:
  - Shift
  - Reduce
  - Left-arc
  - Right-arc
- Advantage: not strictly bottom-up, can create arcs earlier than in the arc-standard model
- The model that you will implement in assignment 3!

Arc-eager model - transitions

- shift:

  $(\sigma,[i|\beta],A) \Rightarrow ([\sigma|i],\beta,A)$

- reduce:

  $([\sigma|i],B,A) \Rightarrow (\sigma,B,A)$ if $(k,l',i) \in A$

- left-arc: if $(k,l',i) \notin A$

  $([\sigma|i],[j|\beta],A) \Rightarrow (\sigma, [j|\beta],A\cup\{j,l,i\})$ and $i \neq 0$

- right-arc:

  $([\sigma|i], [j|\beta],A) \Rightarrow ([\sigma|i|j], \beta,A\cup\{i,l,j\})$

Arc-eager model - oracle

Algorithm 1 Standard oracle for arc-eager dependency parsing

if $c = (\sigma|i, j|\beta, A)$ and $(j, l, i) \in A_{\text{gold}}$ **then**
    $t \leftarrow$ LEFT-ARC$_l$
else if $c = (\sigma|i, j|\beta, A)$ and $(i, l, j) \in A_{\text{gold}}$ **then**
    $t \leftarrow$ RIGHT-ARC$_l$
else if $c = (\sigma|i, j|\beta, A)$ and $\exists k[k < i \wedge \exists l[(k, l, j) \in A_{\text{gold}} \vee (j, l, k) \in A_{\text{gold}}]]$ **then**
    $t \leftarrow$ REDUCE
else
    $t \leftarrow$ SHIFT
return t

- From Goldberg & Nivre, CoLING 2012
- A Dynamic Oracle for Arc-Eager Dependency Parsing

Non-projective transition model

- Allows non-projective parsing by adding a swap transition to the arc-standard model
- Contains four transitions:
  - Shift
  - Swap
  - Left-arc
  - Right-arc
- Runtime is $O(n^2)$ in the worst case (but usually less in practice)

Static oracles

- The ”guide” is a static oracle:
- Two issues:
  - Spurious ambiguity not captured
  - It is never trained on non-gold configurations (at test-time, errors will happen, which will lead to configurations not matching the gold configurations)
- Solution: dynamic oracles!

Dynamic oracles

Algorithm 3 Online training with exploration for greedy transition-based parsers (ith iteration)

for sentence W with gold tree T in corpus **do**
    $c \leftarrow \text{INITIAL}(W)$
    while not $\text{TERMINAL}(c)$ **do**
        $\text{CORRECT}(c) \leftarrow \{t | o(t; c, T) = \textbf{true}\}$
        $t_p \leftarrow \arg\max_{t \in \text{LEGAL}(c)} \mathbf{w} \cdot \phi(c, t)$
        $t_o \leftarrow \arg\max_{t \in \text{CORRECT}(c)} \mathbf{w} \cdot \phi(c, t)$
        if $t_p \notin \text{CORRECT}(c)$ **then**
            $\text{UPDATE}(\mathbf{w}, \phi(c, t_o), \phi(c, t_p))$
            $c \leftarrow \text{EXPLORE}_{k,p}(c, t_o, t_p, i)$
        else
            $c \leftarrow t_p(c)$

function $\text{EXPLORE}_{k,p}(c, t_o, t_p, i)$
    if $i > k$ **and** $\text{RAND}() < p$ **then**
        return $t_p(c)$
    else
        return $\text{NEXT}(c, t_o)$

Figure : Figure taken from Goldberg and Nivre (2013)

Dynamic oracles - cost function, arc hybrid

- $\mathcal{C}(\text{LEFT}; c, T)$: Adding the arc (b, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire heads from $H = \{s_1\} \cup \beta$ and will not be able to acquire dependents from $D = \{b\} \cup \beta$. The cost is therefore the number of arcs in T of the form (s_0, d) and (h, s_0) for $h \in H$ and $d \in D$.

- $\mathcal{C}(\text{RIGHT}; c, T)$: Adding the arc (s_1, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire heads or dependents from $B = \{b\} \cup \beta$. The cost is therefore the number of arcs in T of the form (s_0, d) and (h, s_0) for $h, d \in B$.

- $\mathcal{C}(\text{SHIFT}; c, T)$: Pushing b onto the stack means that b will not be able to acquire heads from $H = \{s_1\} \cup \sigma$, and will not be able to acquire dependents from $D = \{s_0, s_1\} \cup \sigma$. The cost is therefore the number of arcs in T of the form (b, d) and (h, b) for $h \in H$ and $d \in D$.

Other alternatives

- Parsing with beam search
  - Instead of just keeping the 1-best tree, keep a beam of the k-best trees in each step
  - Requires scoring and ranking of transition sequences
  - Complexity: O(nk)

UPPSALA UNIVERSITET

Non-projective dependency parsing

- Variants of transition-based parsing
  - Using a swap-transition
- Graph-based parsing
  - Minimum spanning tree algorithms
- Post processing
  - Pseudo-projective parsing
  - Approximate non-projective parsing

UPPSALA
UNIVERSITET

Cross-lingual parsing

- Popular in recent research
- Main purpose: improve parsing performance for a low-resource language by using data from other (related) language(s)
  - Zero-shot
  - Few-shot
  - Joint training / polyglot models
- Two main approaches:
  - Annotation transfer
  - Model transfer

UPPSALA UNIVERSITET

The end of the course

- Literature seminar 2, **Wednesday March 1**
- Assignments
  - Scheduled supervision
- Project
  - Contact Sara for help if needed
- Plan your workload carefully!

Project

- Proposal: February 27
  - Sign up to groups in Studium, 1-2 students
- Supervision on demand
- Final report: March 24
- Final seminar, March 22
  - No formal presentation with slides
  - Be prepared to describe and discuss your project in smaller groups
  - Both participants in a pair project should be able to discuss the project independently