



UPPSALA  
UNIVERSITET

# Collins' and Eisner's algorithms

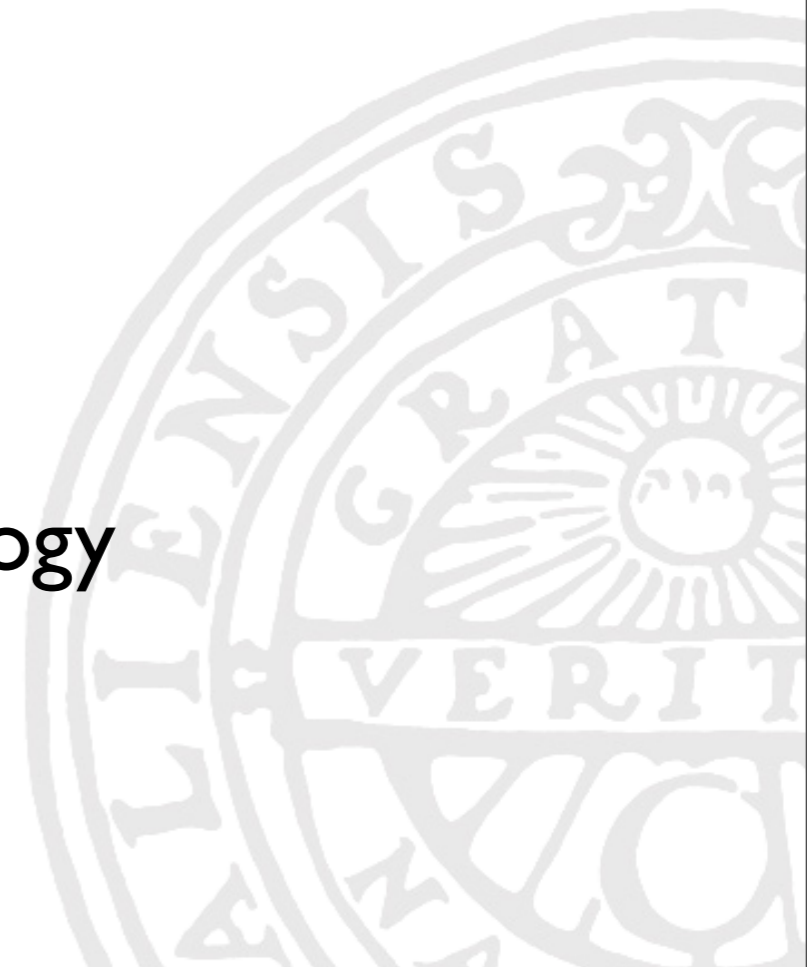
Syntactic analysis/parsing

2017-12-07

Sara Stymne

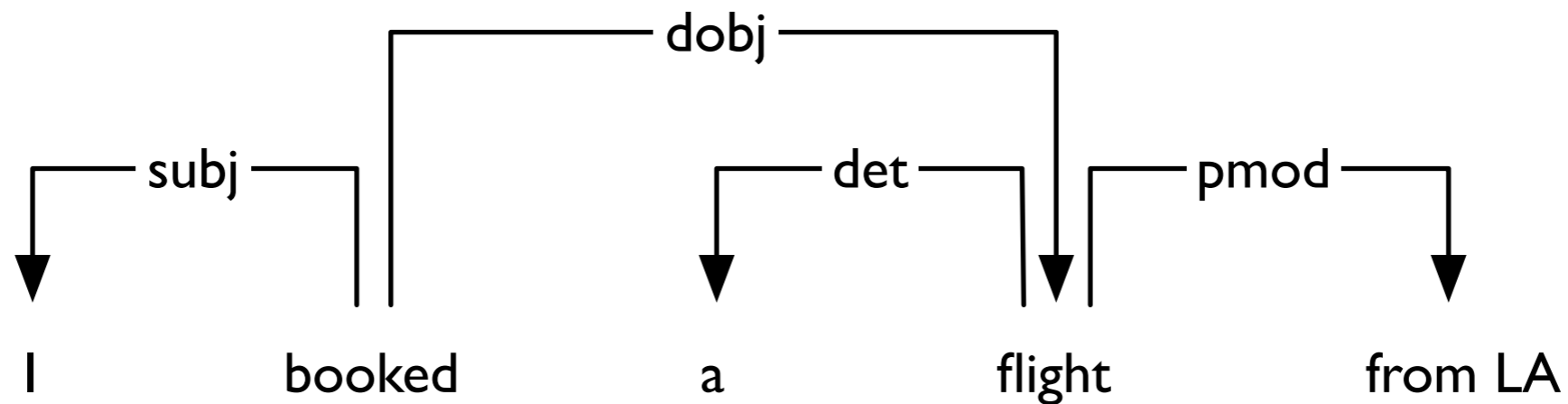
Department of Linguistics and Philology

Based on slides from Marco Kuhlmann





# Recap: Dependency trees



- In an arc  $h \rightarrow d$ , the word  $h$  is called the **head**, and the word  $d$  is called the **dependent**.
- The arcs form a rooted tree.



# Recap: Scoring models and parsing algorithms

Distinguish two aspects:

- **Scoring model:**

How do we want to score dependency trees?

- **Parsing algorithm:**

How do we compute a highest-scoring dependency tree under the given scoring model?



# Recap: The arc-factored model

- To score a dependency tree, score the individual arcs, and combine the score into a simple sum.

$$\text{score}(t) = \text{score}(a_1) + \dots + \text{score}(a_n)$$

- Define the **score** of an arc  $h \rightarrow d$  as the weighted sum of all features of that arc:

$$\text{score}(h \rightarrow d) = f_1 w_1 + \dots + f_n w_n$$



# Recap: Example features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
- ‘The head is a verb  
*and* the dependent is a noun.’
- ‘The head is a verb  
*and* the predecessor of the head is a pronoun.’
- ‘The arc goes from left to right.’
- ‘The arc has length 2.’



# Recap: Training using structured prediction

- Take a sentence  $w$  and a gold-standard dependency tree  $g$  for  $w$ .
- Compute the highest-scoring dependency tree under the current weights; call it  $p$ .
- Increase the weights of all features that are in  $g$  but not in  $p$ .
- Decrease the weights of all features that are in  $p$  but not in  $g$ .

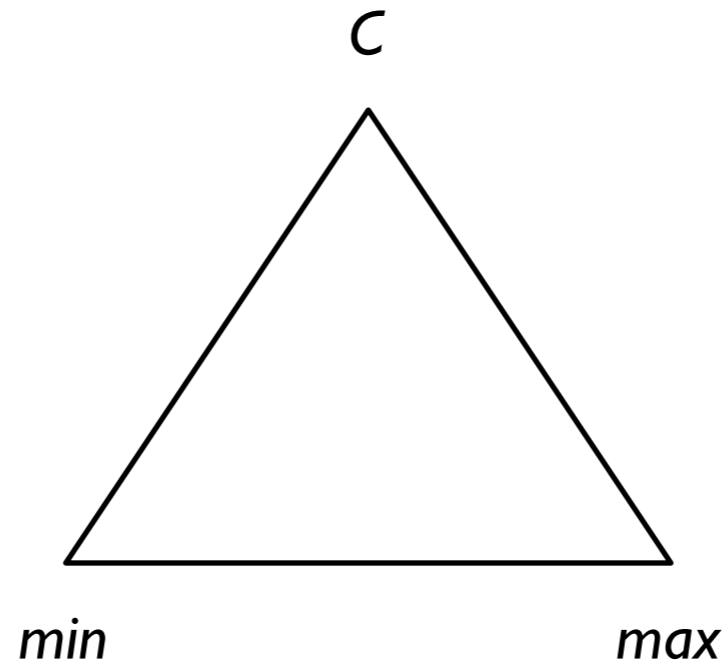


# Recap: Collin's algorithm

- Collin's algorithm is a simple algorithm for computing the highest-scoring dependency tree under an arc-factored scoring model.
- It can be understood as an extension of the CKY algorithm to dependency parsing.
- Like the CKY algorithm, it can be characterized as a bottom-up algorithm based on dynamic programming.



# Recap: Signatures

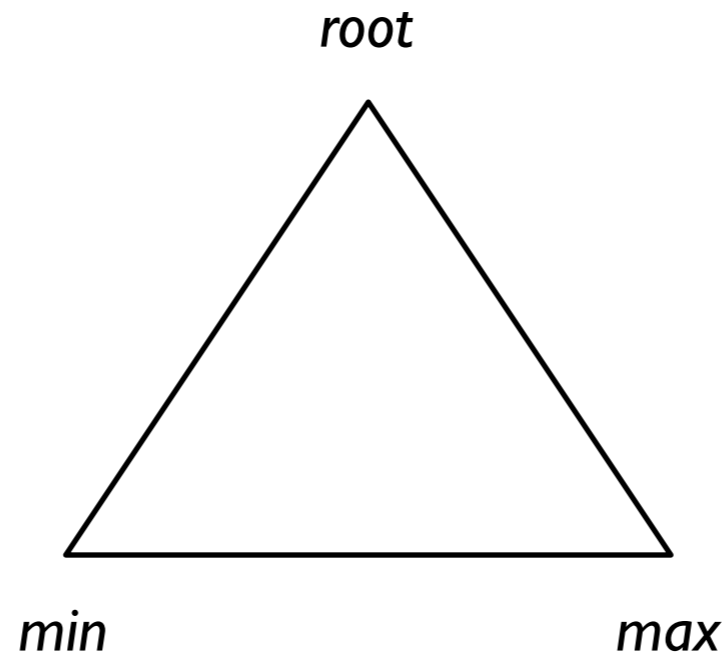


$[min, max, c]$





# Recap: Signatures



**[*min, max, root*]**



UPPSALA  
UNIVERSITET

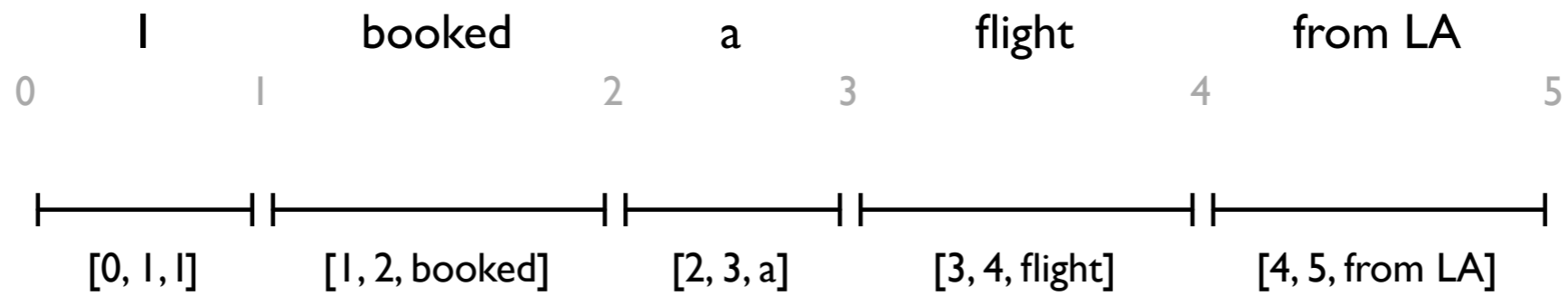
Collins' algorithm

# Recap: Initialization

0    1    2    3    4    5  
I    booked    a    flight    from LA



# Recap: Initialization



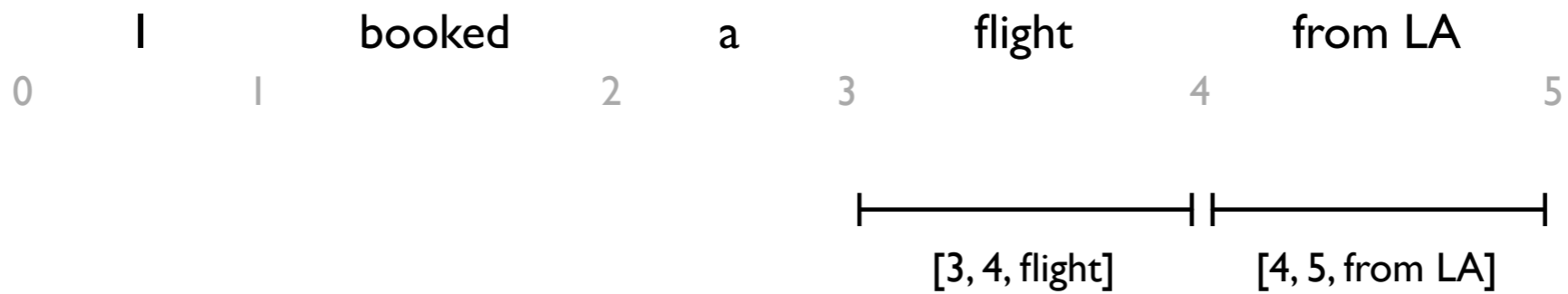


# Recap: Adding a left-to-right arc

0        1        2        3        4        5  
I        booked        a        flight        from LA

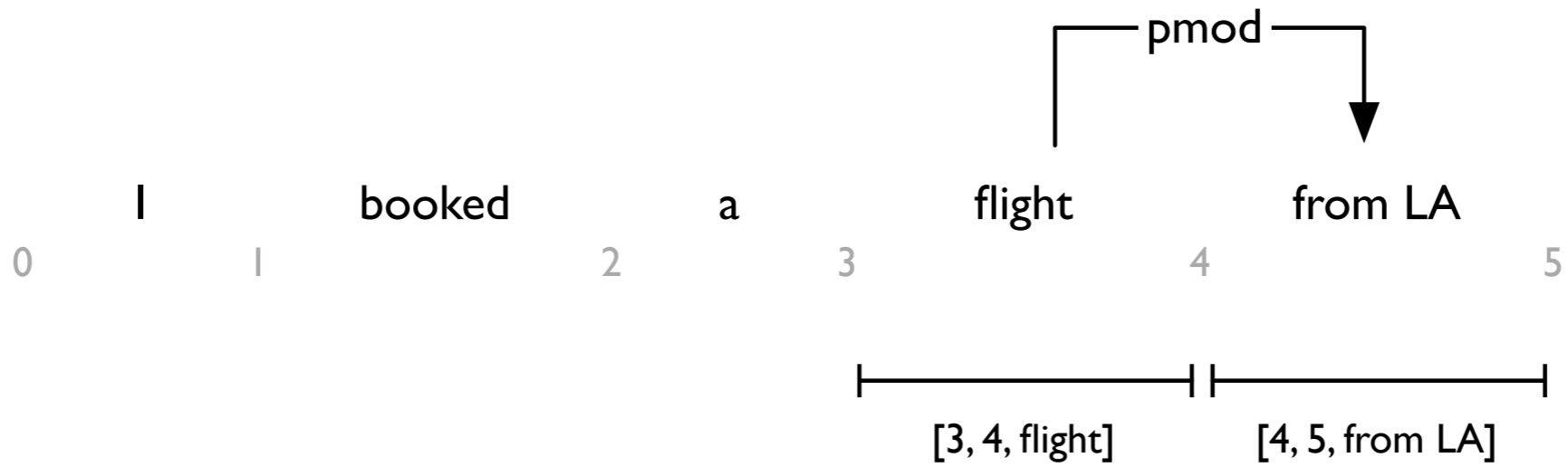


# Recap: Adding a left-to-right arc



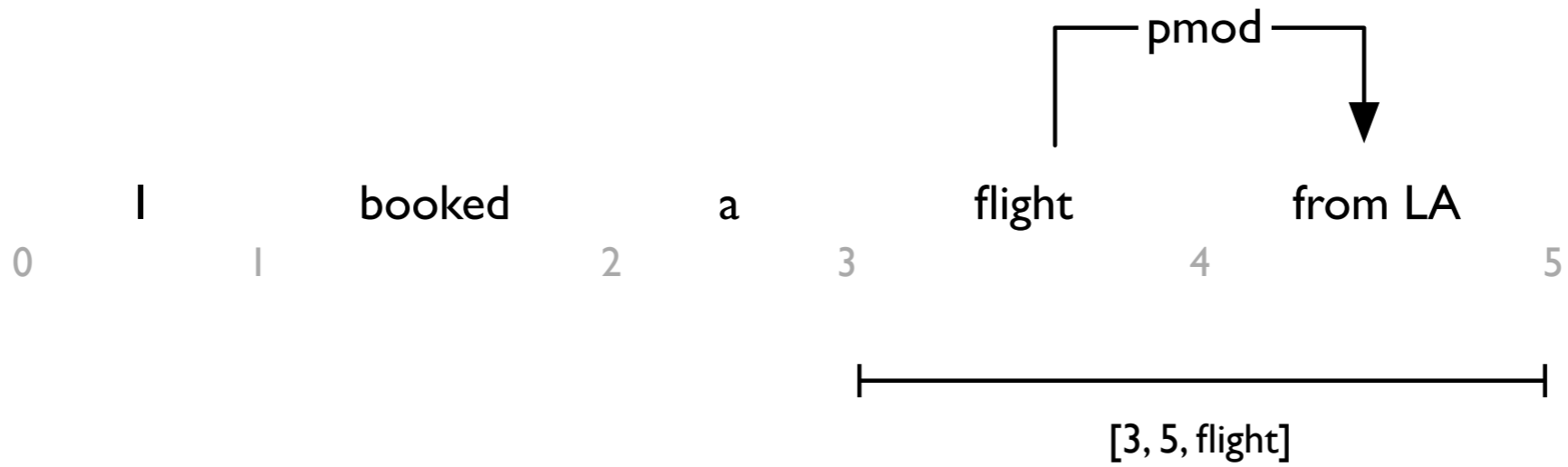


# Recap: Adding a left-to-right arc





# Recap: Adding a left-to-right arc





UPPSALA  
UNIVERSITET

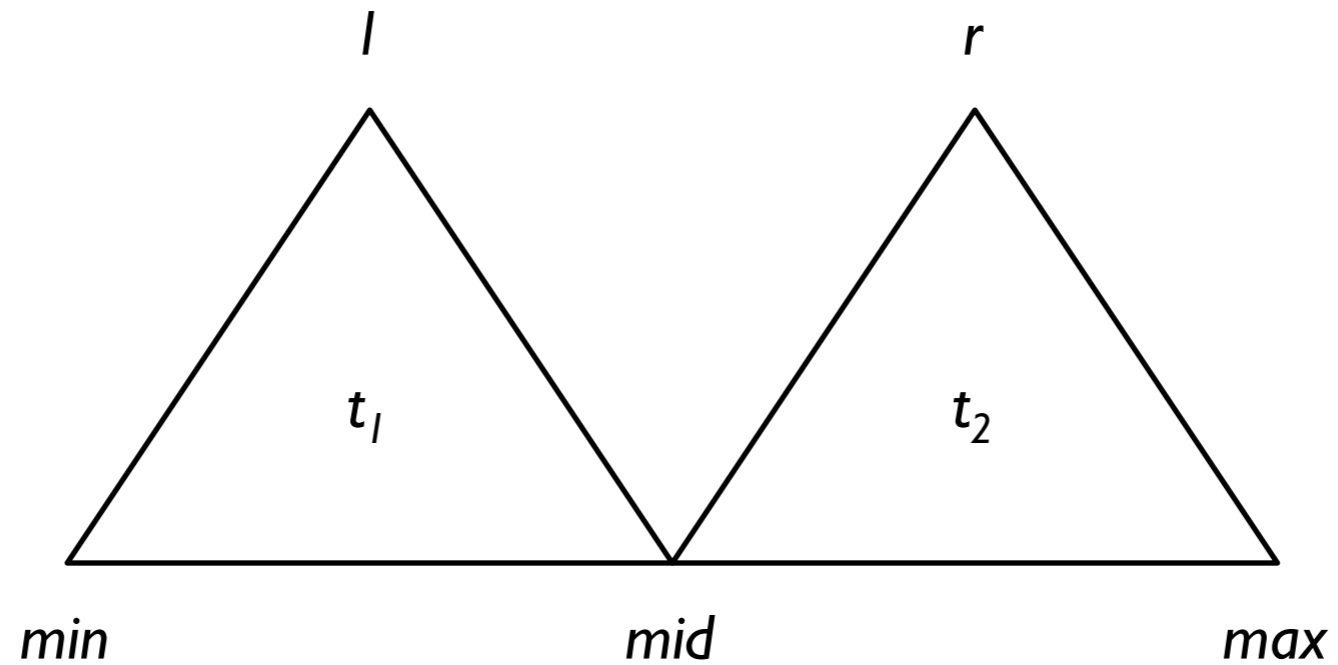
Collins' algorithm

# Recap: Adding a left-to-right arc



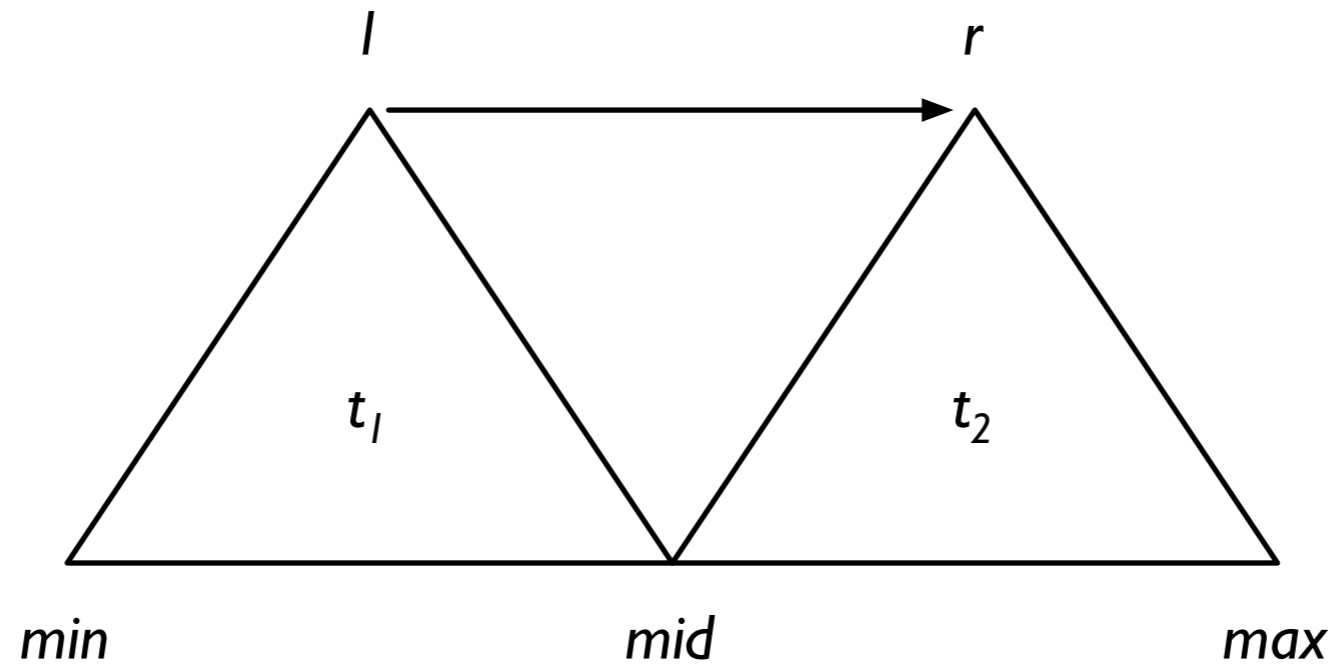


# Recap: Adding a left-to-right arc





# Recap: Adding a left-to-right arc

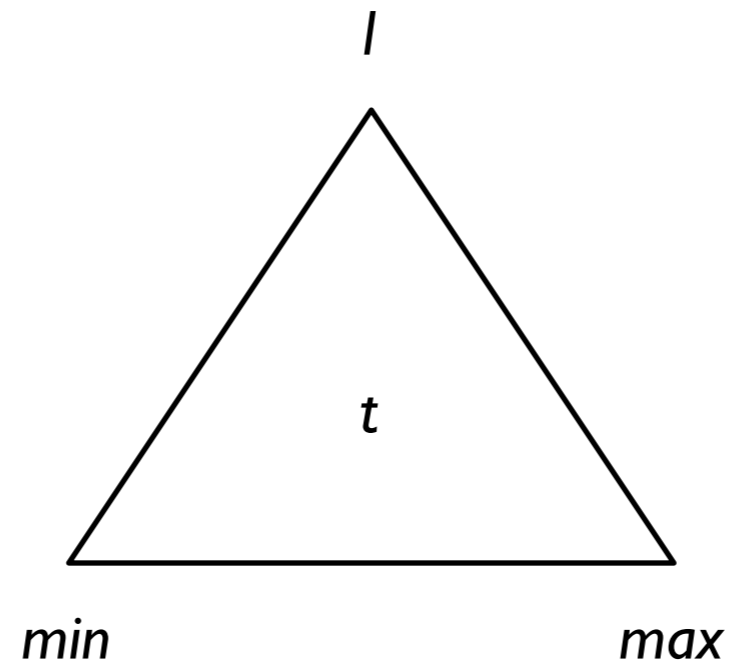




UPPSALA  
UNIVERSITET

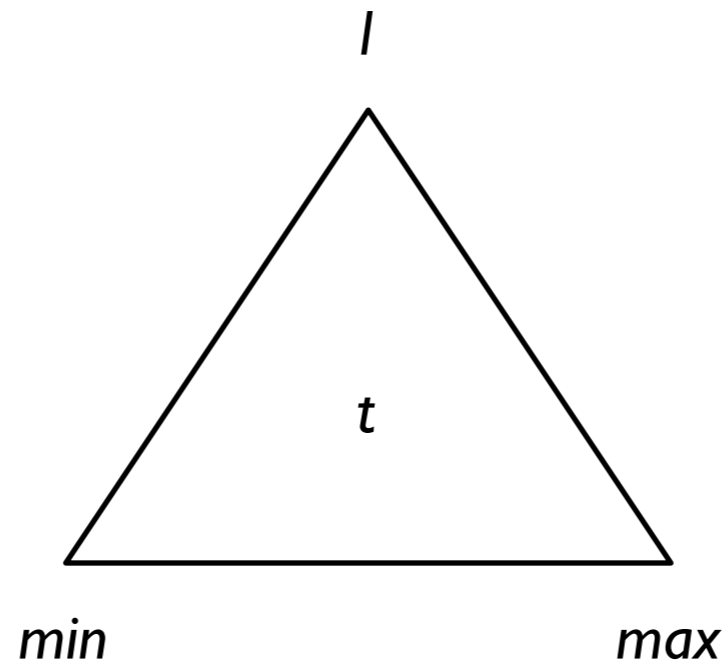
Collins' algorithm

# Recap: Adding a left-to-right arc





# Recap: Adding a left-to-right arc



$$\text{score}(t) = \text{score}(t_1) + \text{score}(t_2) + \text{score}(l \rightarrow r)$$

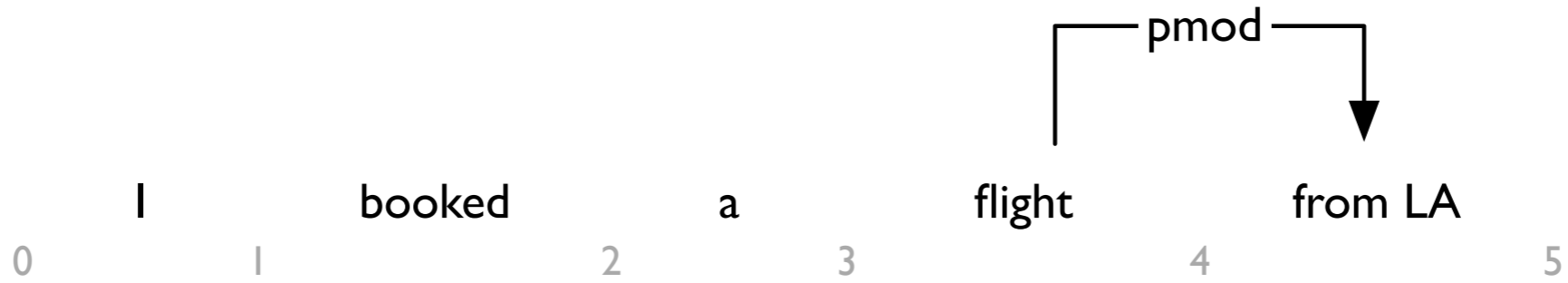


# Adding a left-to-right arc

```
for each [min, max] with max - min > 1 do
  for each l from min to max - 2 do
    double best = score[min][max][l]
    for each r from l + 1 to max - 1 do
      for each mid from l + 1 to r do
        t1 = score[min][mid][l]
        t2 = score[mid][max][r]
        double current = t1 + t2 + score(l → r)
        if current > best then
          best = current
    score[min][max][l] = best
```

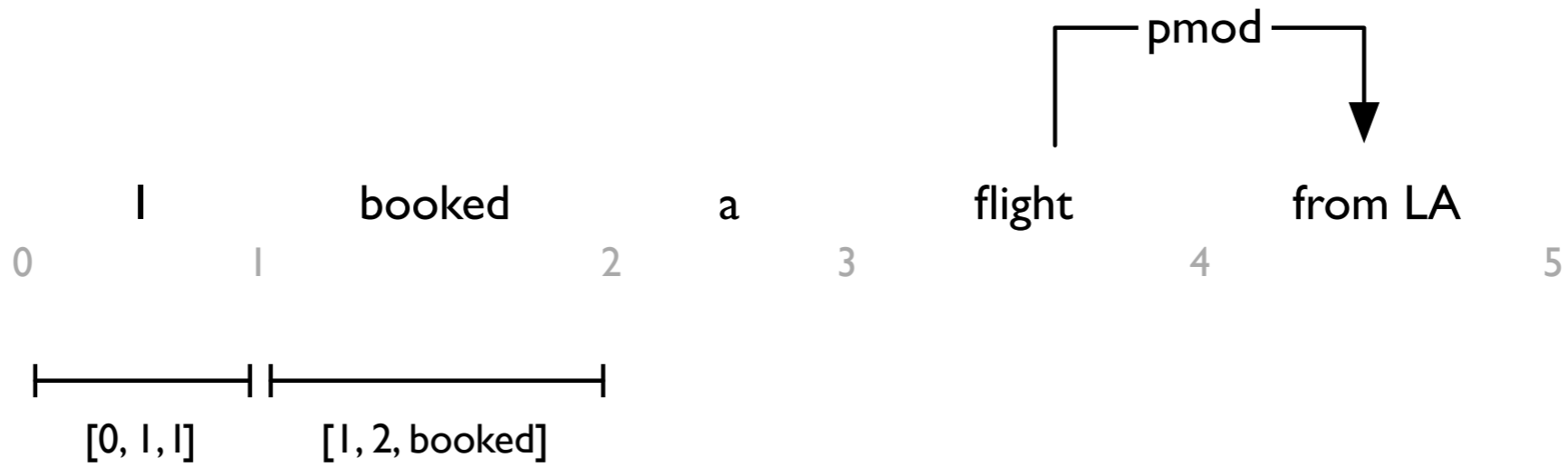


# Adding a right-to-left arc



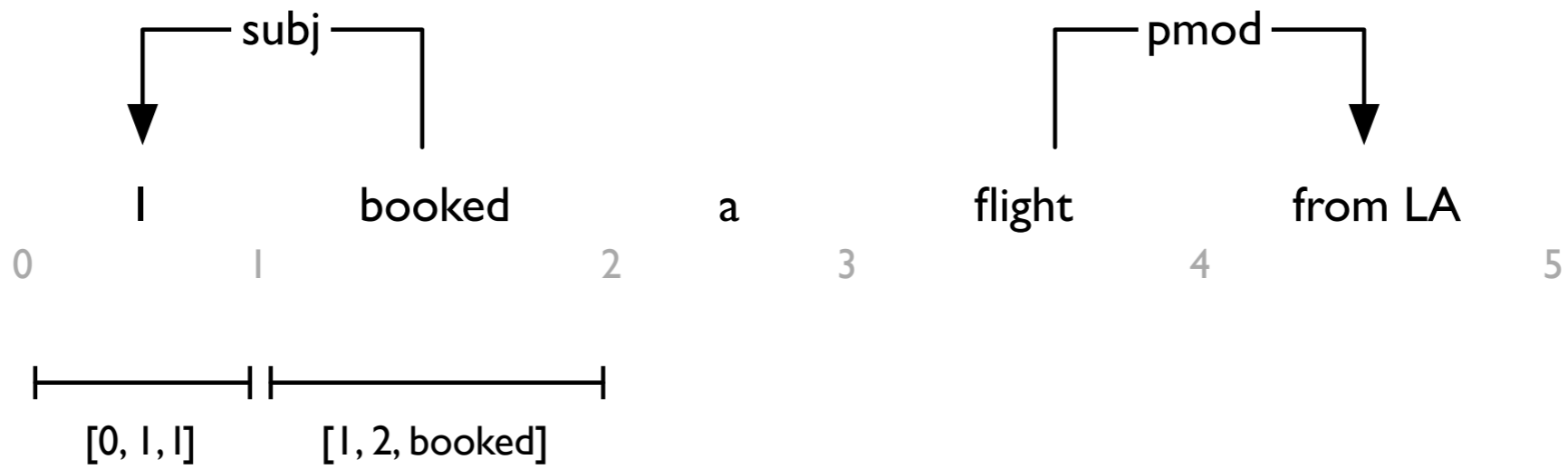


# Adding a right-to-left arc





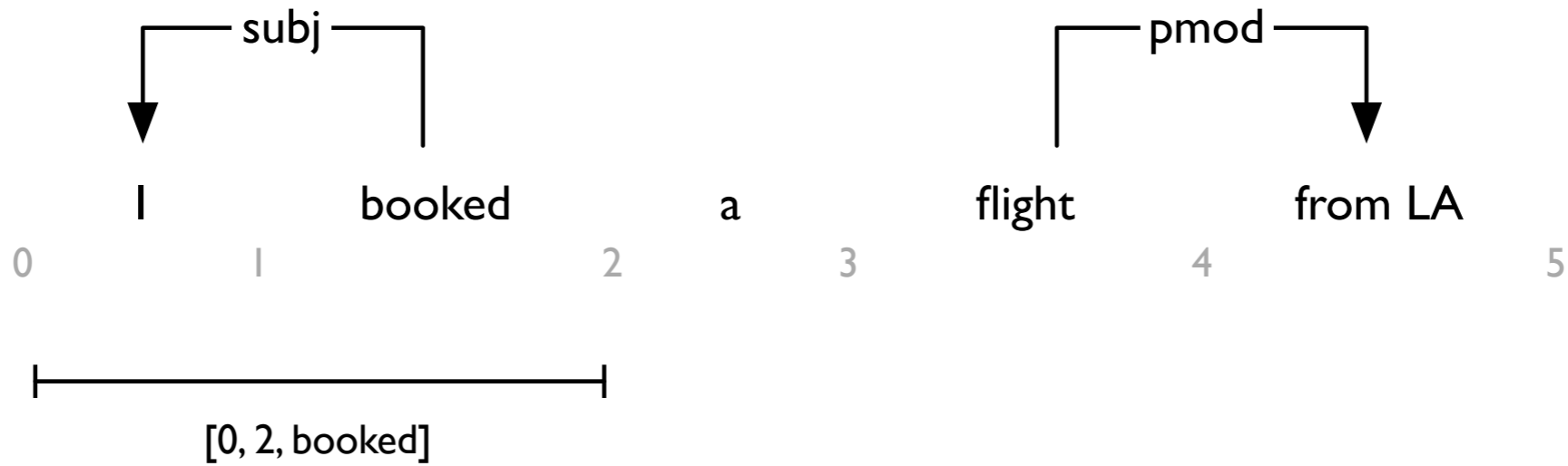
# Adding a right-to-left arc





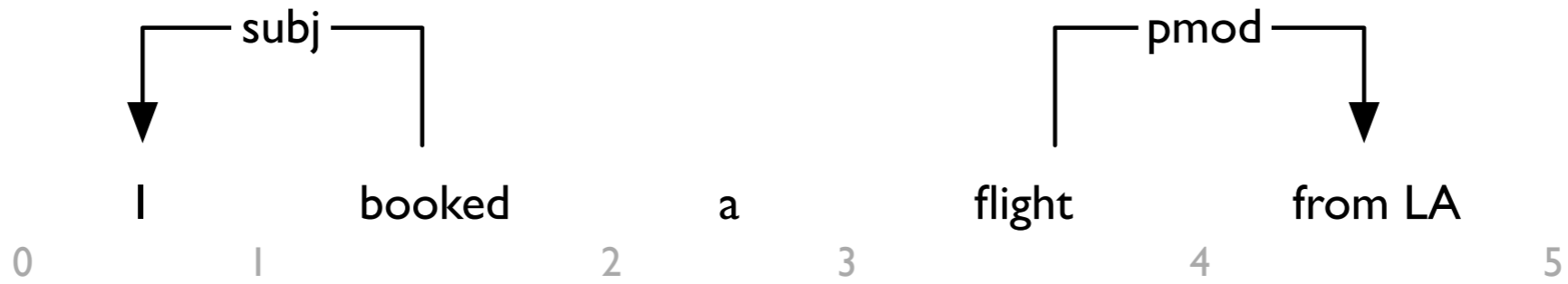


# Adding a right-to-left arc



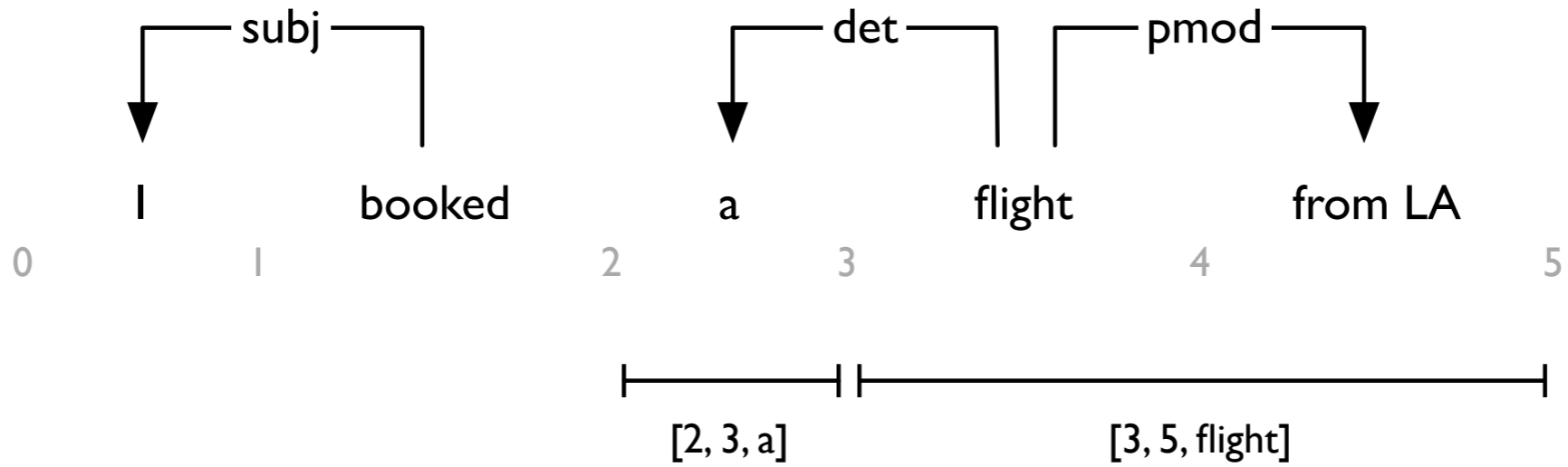


# Finishing up



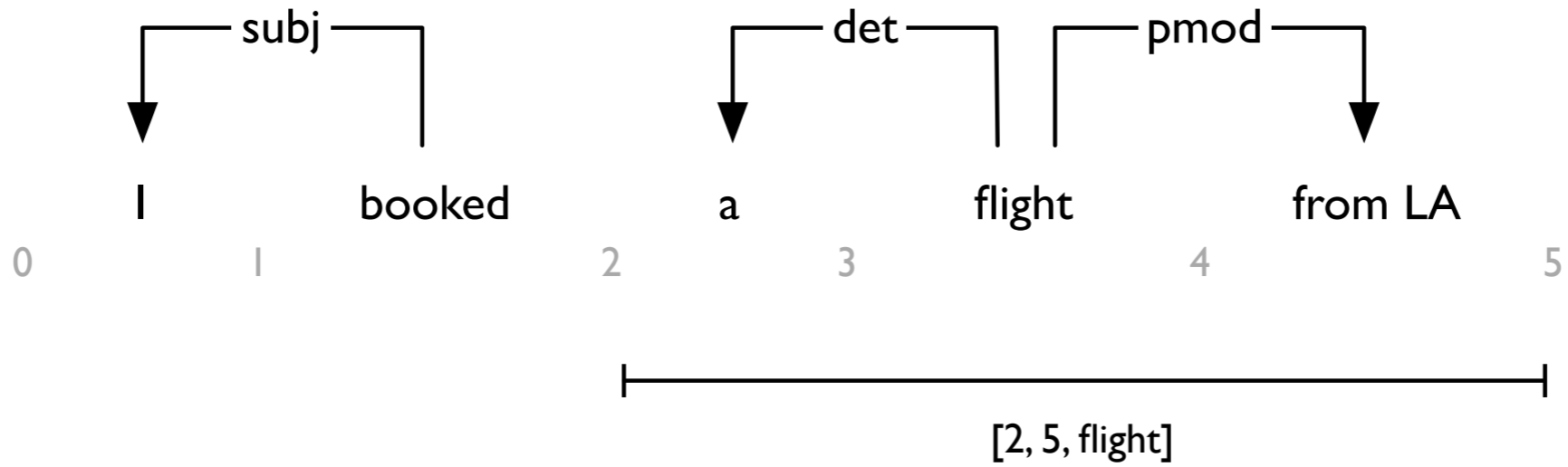


# Finishing up



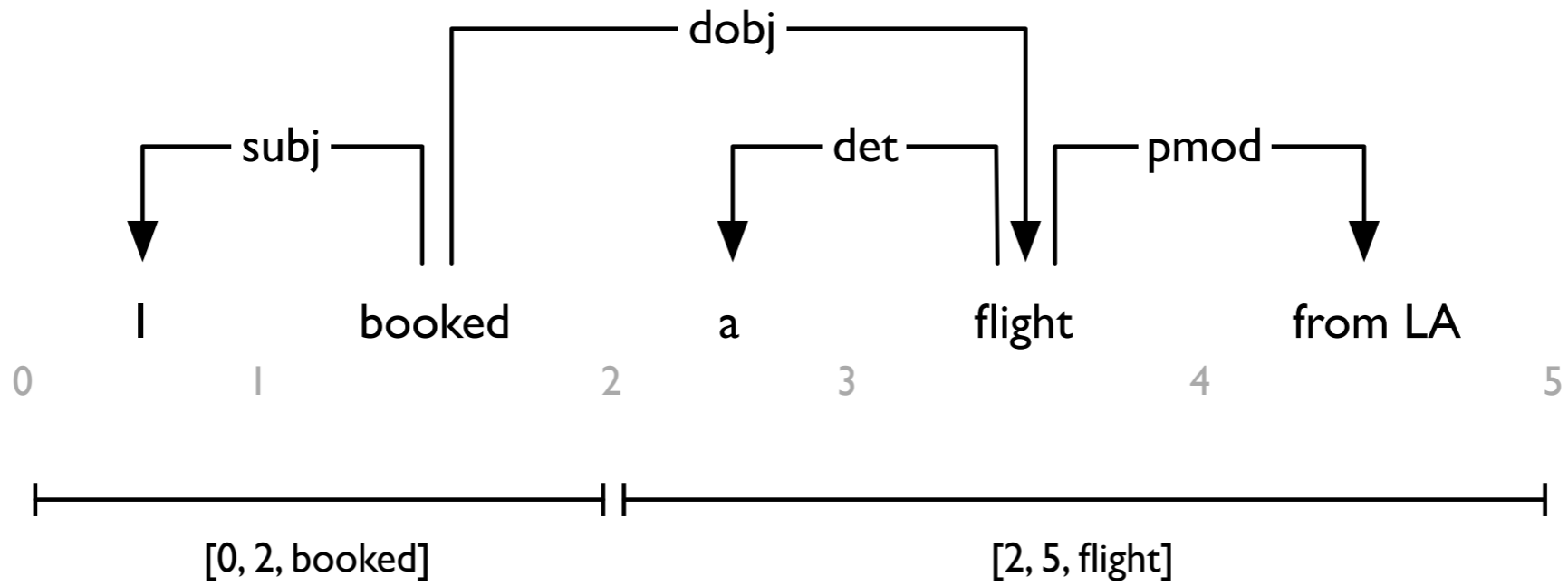


# Finishing up



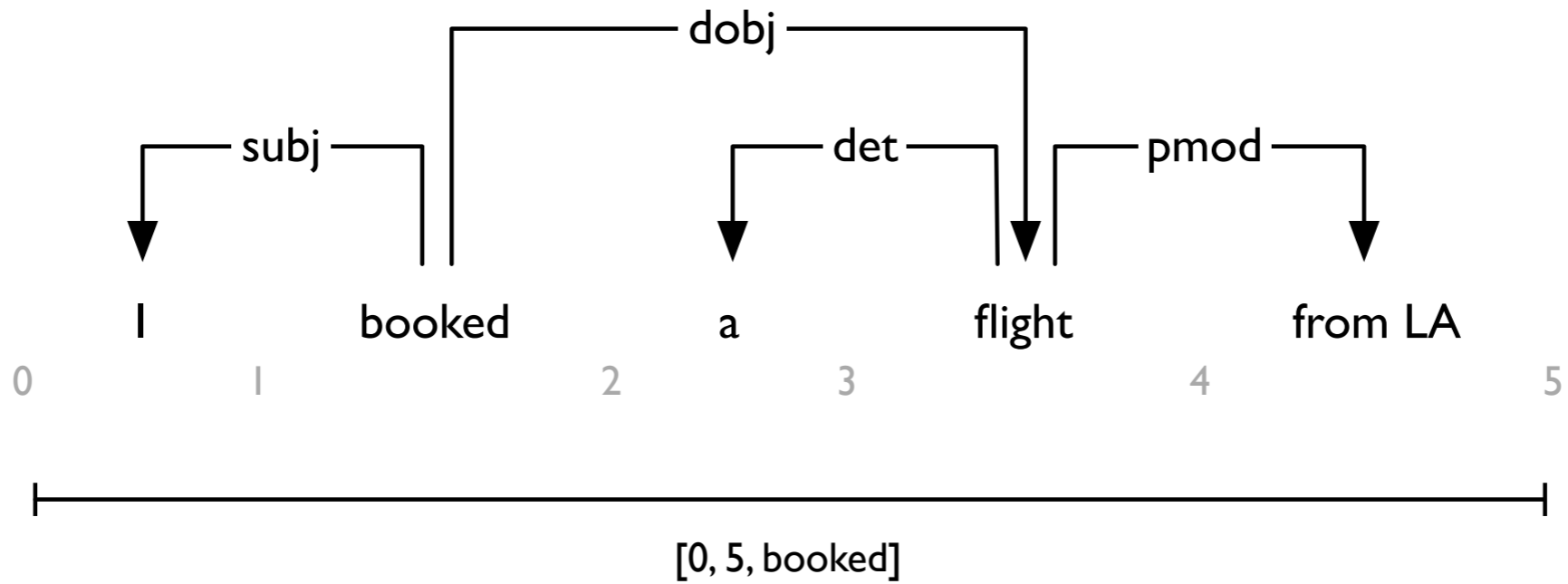


# Finishing up





# Finishing up





# Recap: Complexity analysis

- **Space requirement:**  
 $O(|w|^3)$
- **Runtime requirement:**  
 $O(|w|^5)$



## Extension to the labeled case

- It is important to distinguish dependencies of different types between the same two words.

*Example:* subj, dobj

- For this reason, practical systems typically deal with **labeled arcs**.
- The question then arises how to extend Collins' algorithm to the labeled case.





# Naive approach

- Add an innermost loop that iterates over all edge labels in order to find the combination that maximizes the overall score.
- For each step of the original algorithm, we now need to make  $|L|$  steps, where  $L$  is the set of all labels.



## Smart approach

- Before parsing, compute a table that lists, for each head–dependent pair  $(h, d)$ , the label that maximizes the score of arcs  $h \rightarrow d$ .
- During parsing, simply look up the best label in the precomputed table.
- This adds (not multiplies!) a factor of  $|L||w|^2$  to the overall runtime of the algorithm.



# Summary

- Collins' algorithm is a CKY-style algorithm for computing the highest-scoring dependency tree under an arc-factored scoring model.
- It runs in time  $O(|w|^5)$ .  
This may not be practical for long sentences.



UPPSALA  
UNIVERSITET

# Eisner's algorithm

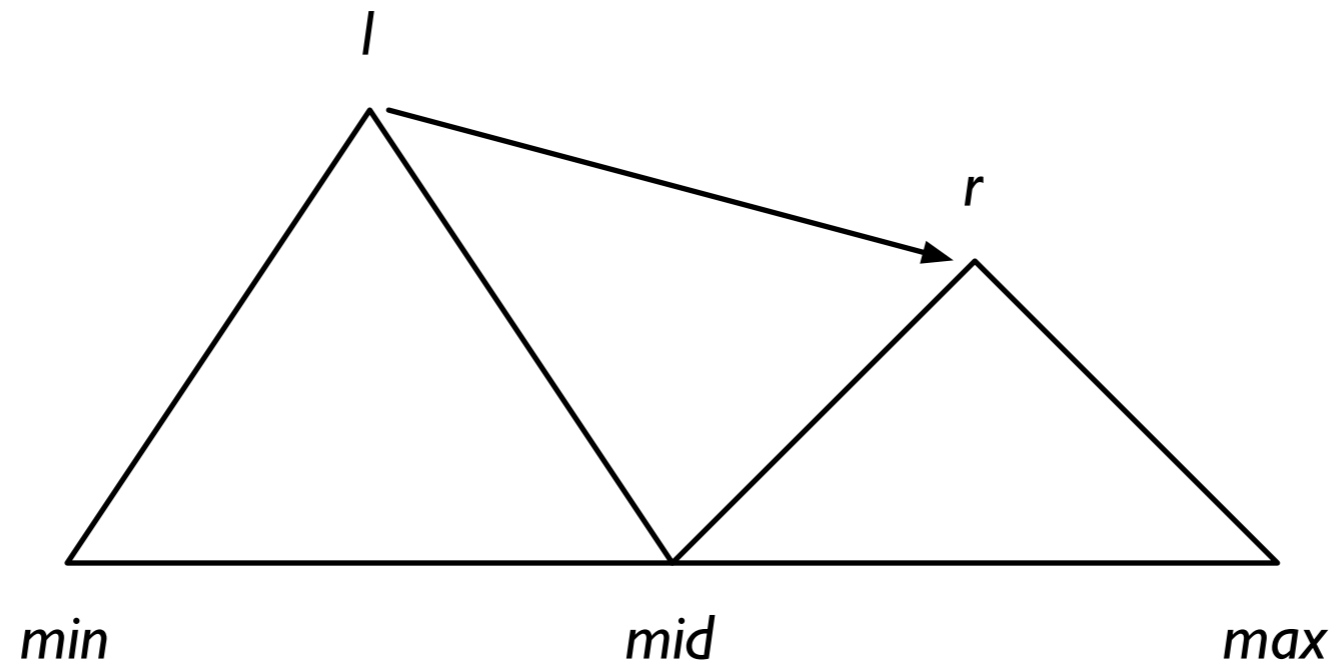


# Eisner's algorithm

- With its runtime of  $O(|w|^5)$ , Collins' algorithm may not be of much use in practice.
- With Eisner's algorithm we will be able to solve the same problem in  $O(|w|^3)$ .
  - Intuition: collect left and right dependents independently



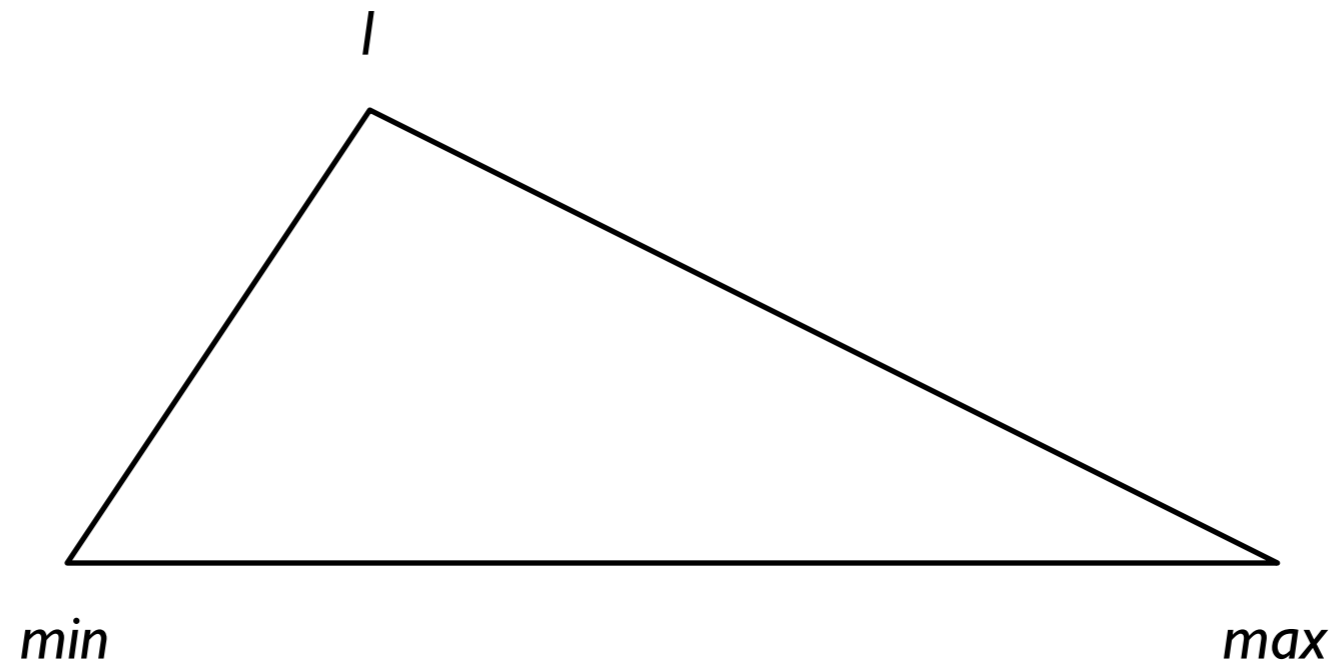
# Basic idea



In Collins' algorithm, adding a left-to-right arc is done in one single step, specified by 5 positions.



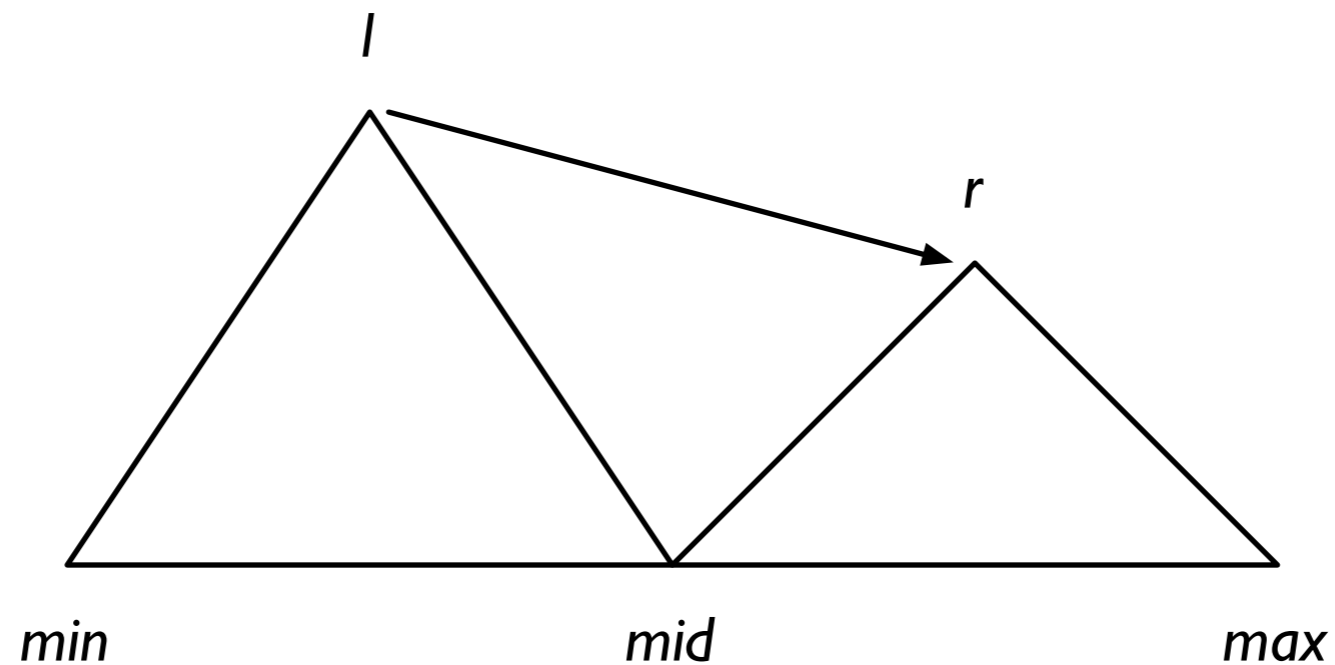
# Basic idea



In Collins' algorithm, adding a left-to-right arc is done in one single step, specified by 5 positions.



# Basic idea

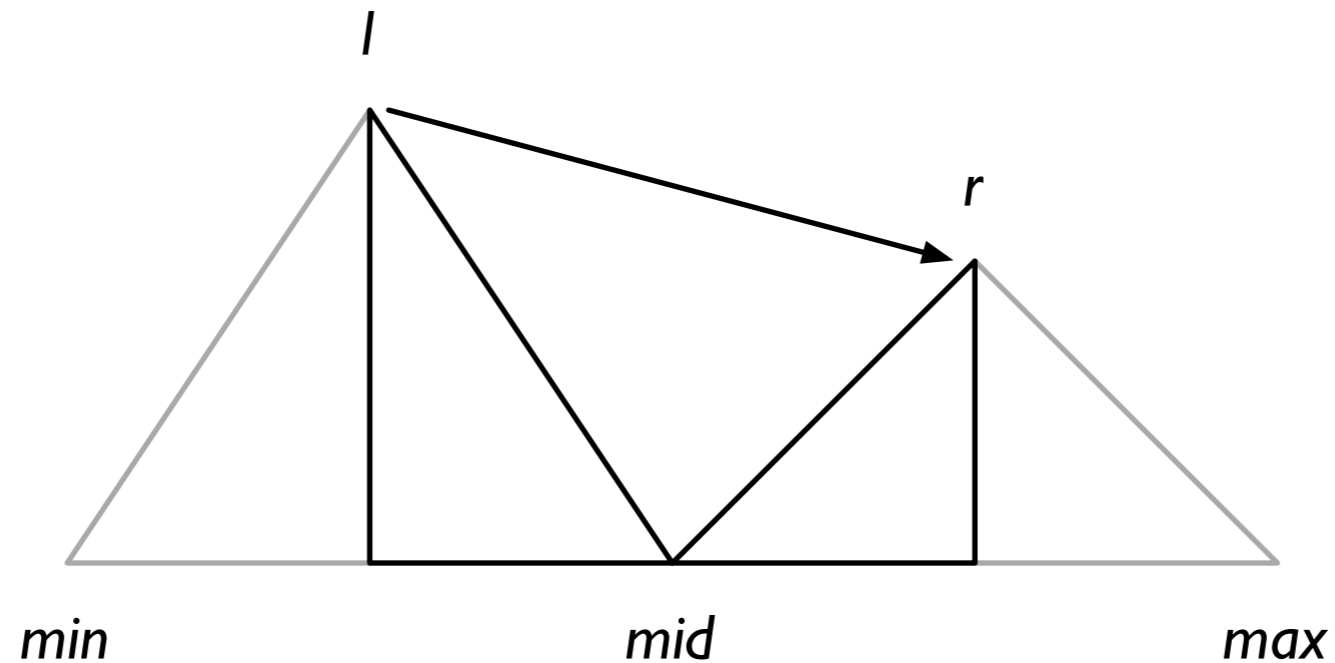


In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.





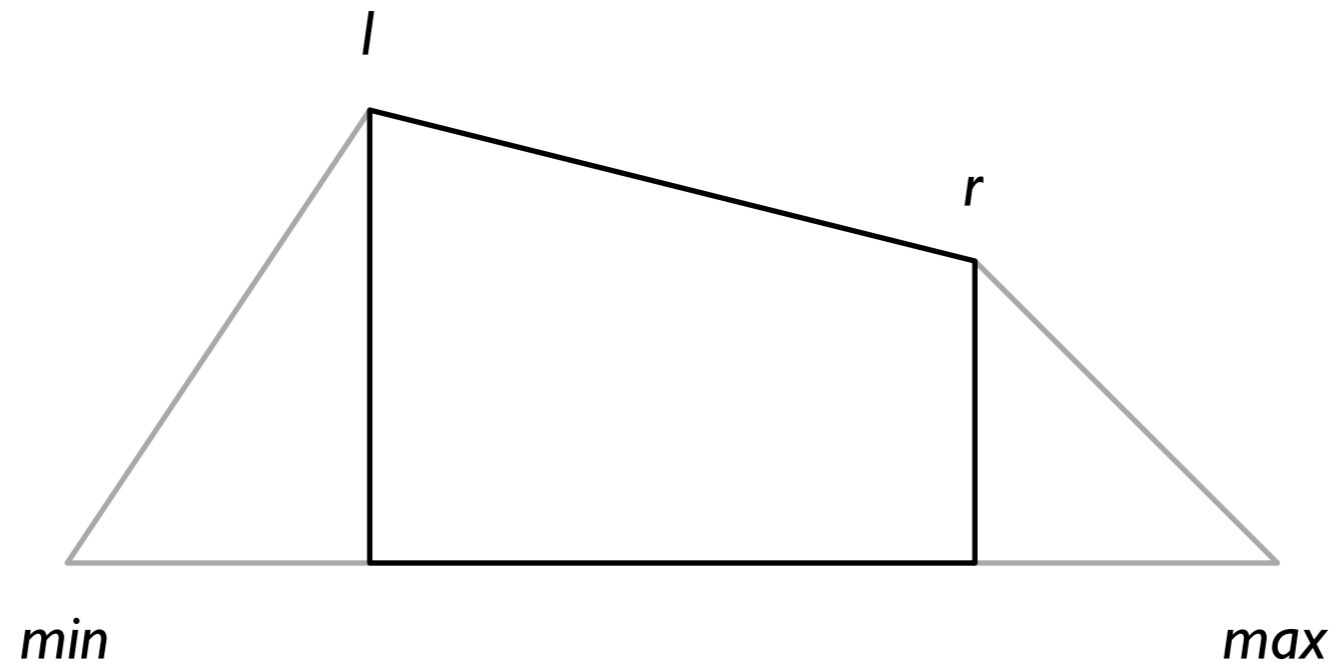
# Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.



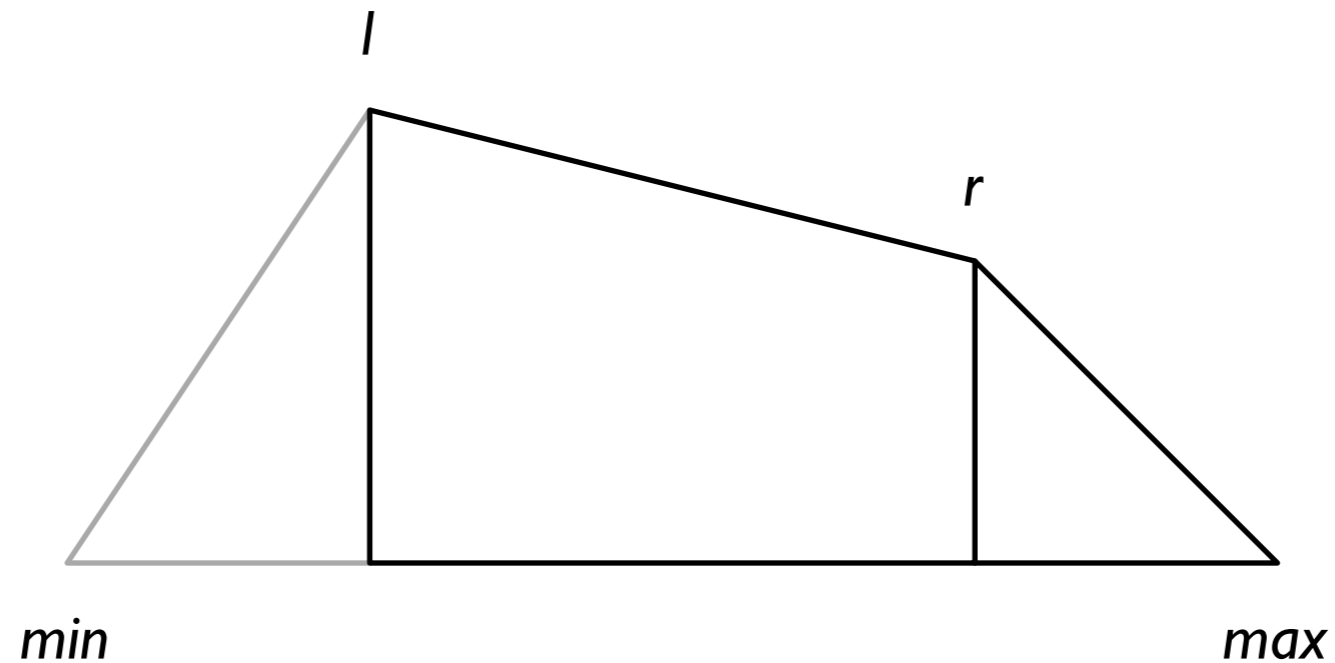
# Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.



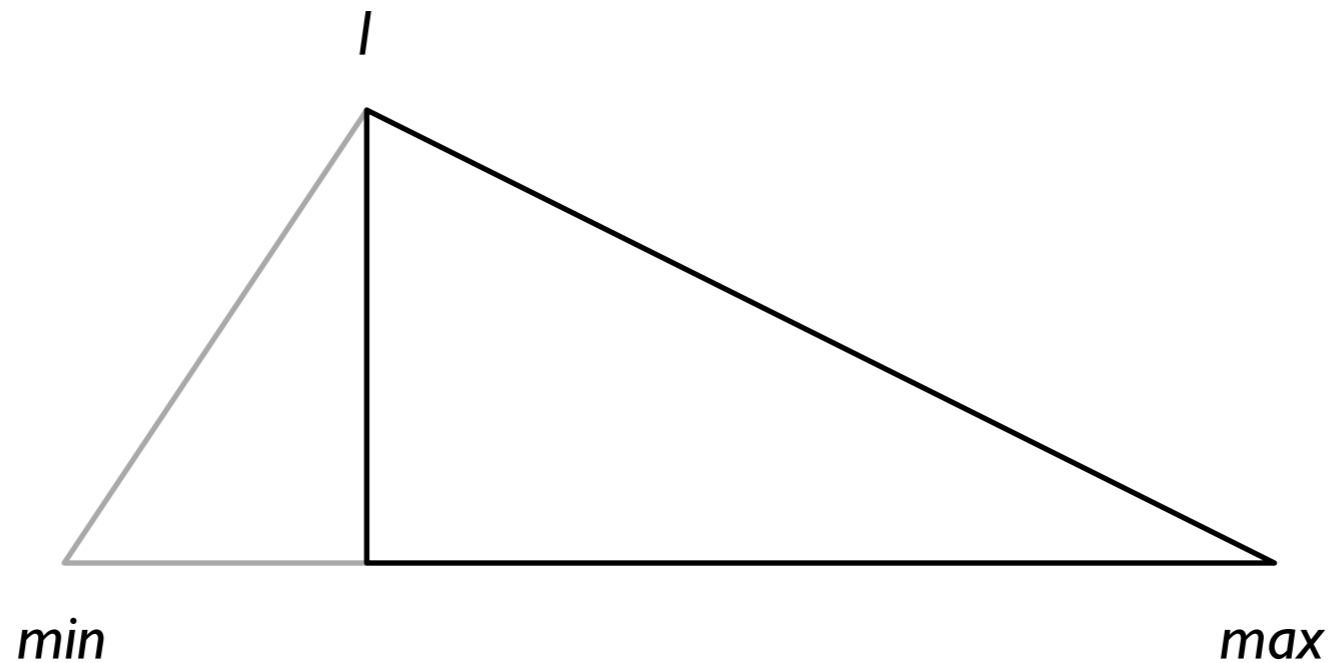
# Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.



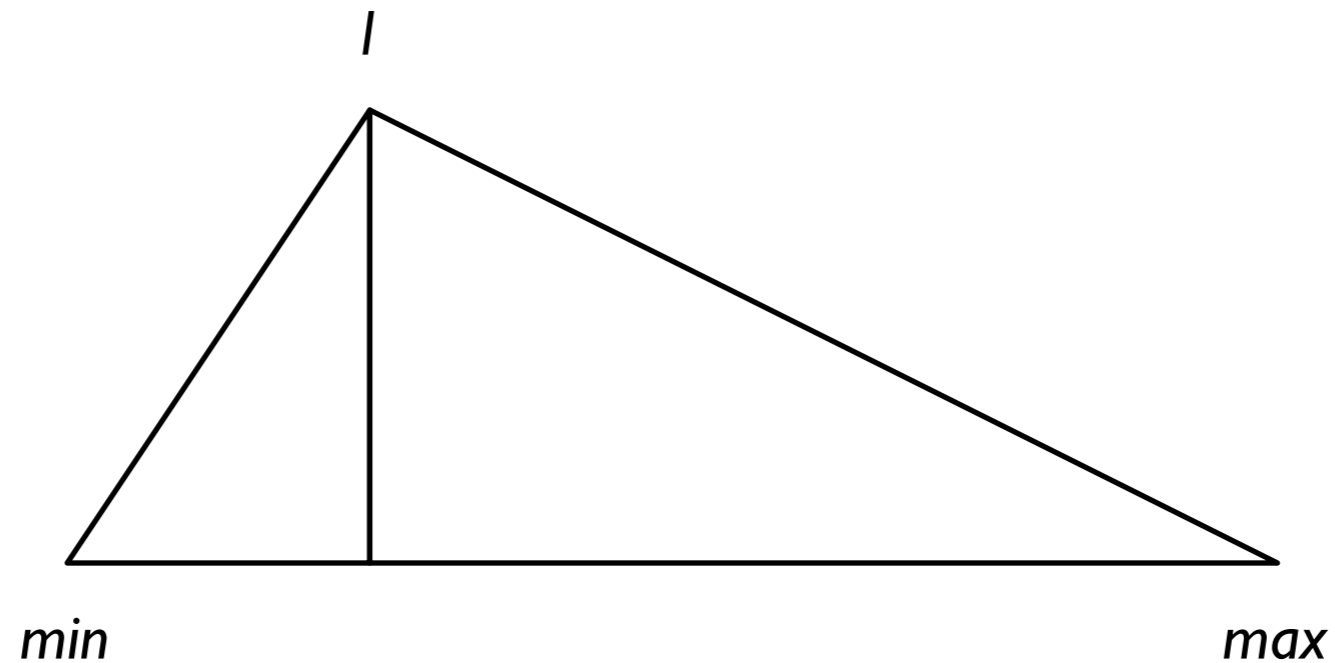
# Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.



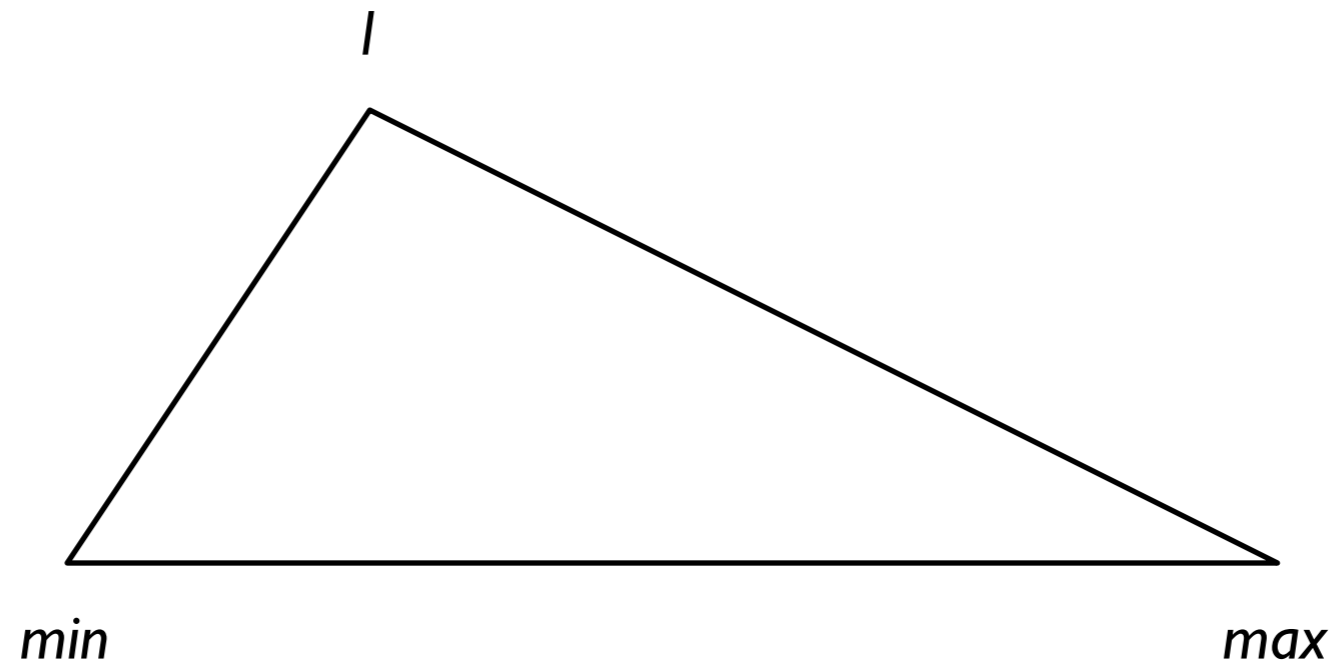
# Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.



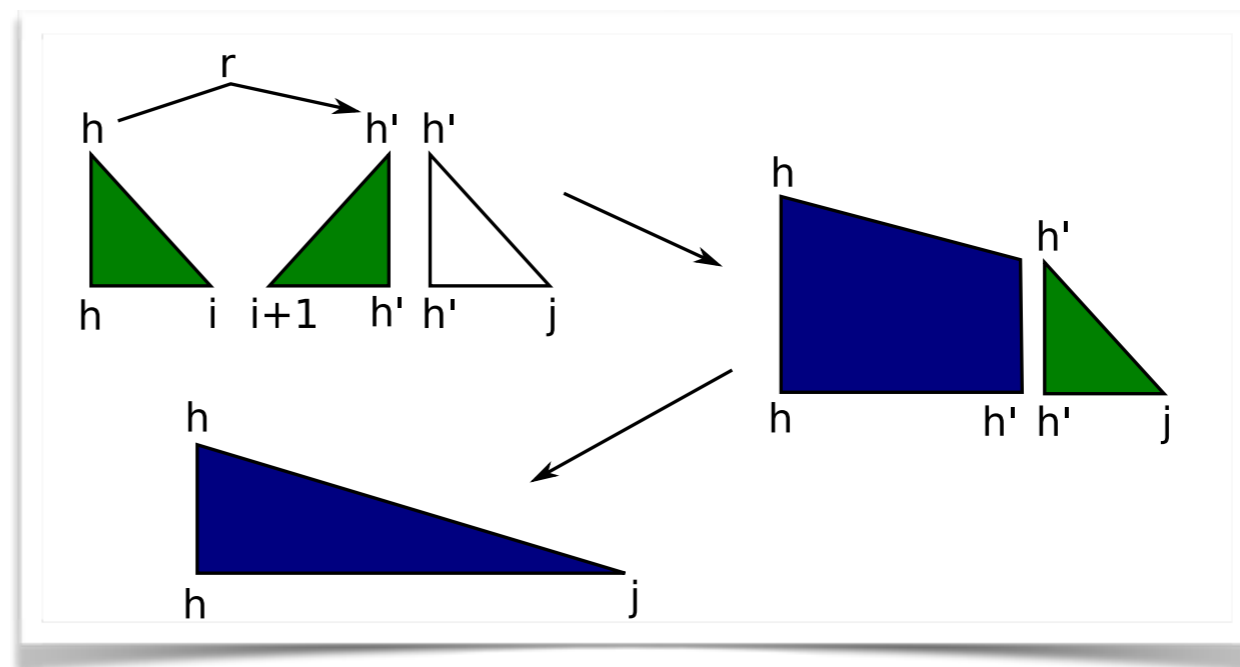
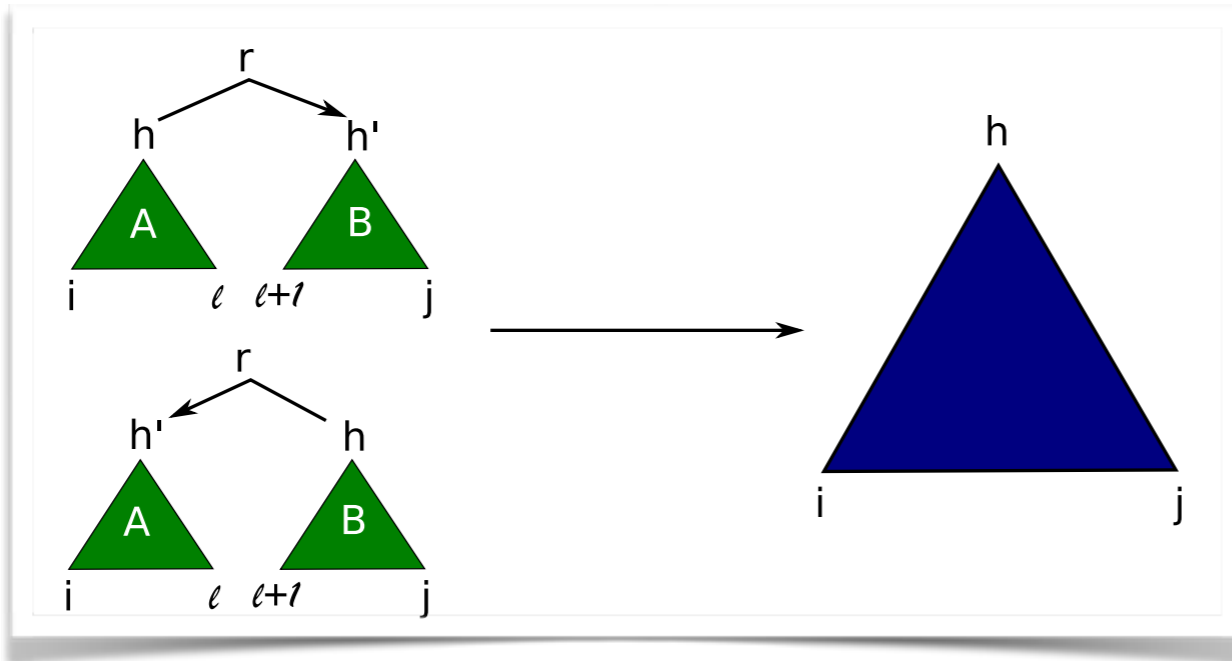
# Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.



# Comparison





# Dynamic programming tables

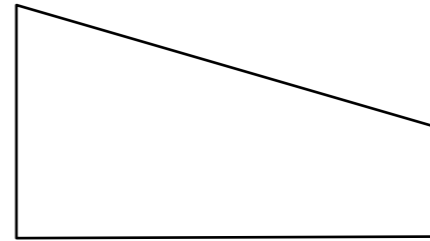
- Collins':
  - [min,max,head]
- Eisner's
  - [min,max,head-side,complete]
    - head-side (binary): is head to the left or right?
    - complete (binary:) is the non-head side still looking for dependents?



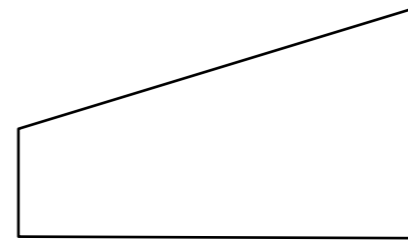


# Graphic representation

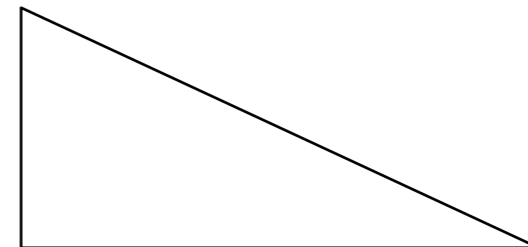
- [min,max,left,yes]



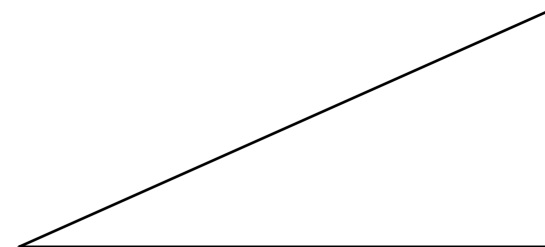
- [min,max,right,yes]



- [min,max,left,no]

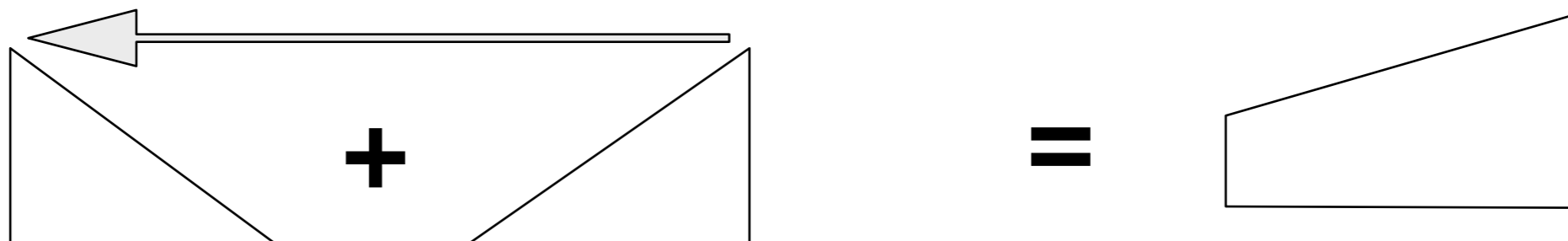
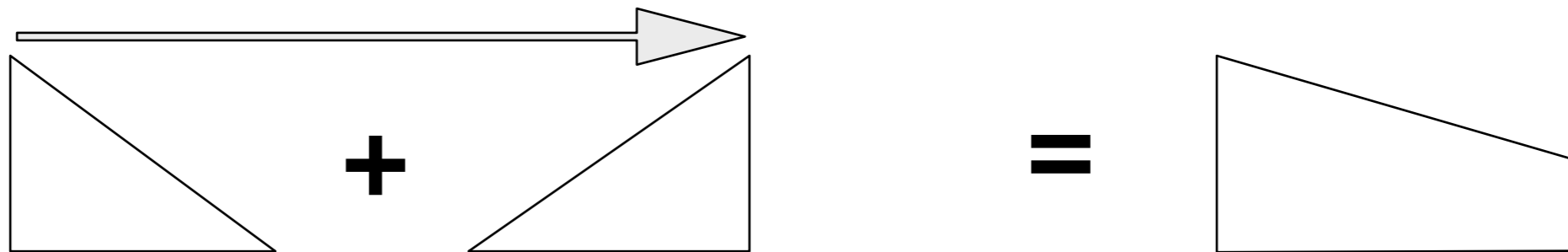
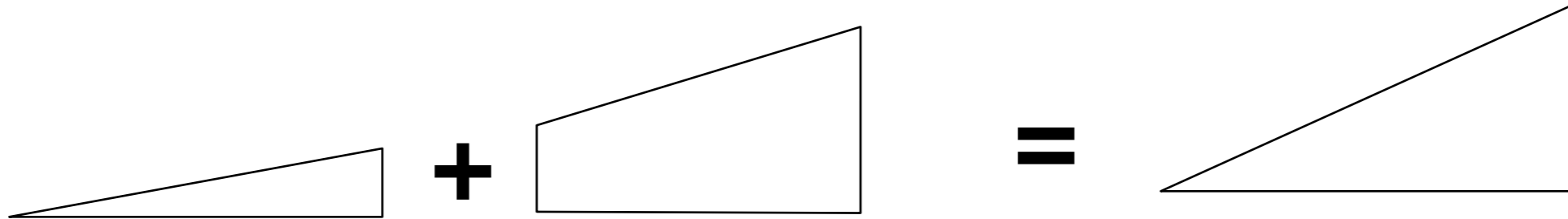
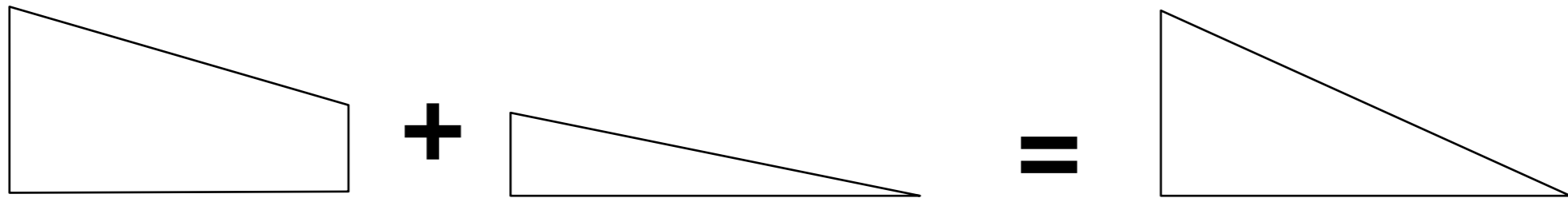


- [min,max,right,no]





# Possible operations





## Pseudo code

```
for each i from 0 to n and all d,c do
    C[i][i][d][c] = 0.0
for each m from 1 to n do
    for each i from 0 to n-m do
        j = i+m
        C[i][j][←][1] = maxi≤q<j(C[i][q][→][0] + C[q+1][j][←][0]+score(wj,wi))
        C[i][j][→][1] = maxi≤q<j(C[i][q][→][0] + C[q+1][j][←][0]+score(wi,wj))
        C[i][j][←][0] = maxi≤q<j(C[i][q][←][0] + C[q][j][←][1])
        C[i][j][→][0] = maxi≤q<j(C[i][q][→][1] + C[q][j][→][0])
return [0][n][→][0]
```



# Summary

- Eisner's algorithm is an improvement over Collin's algorithm that runs in time  $O(|w|^3)$ .
- The same scoring model can be used.
- The same technique for extending the parser to labeled parsing can be used, adding  $O(|L||w|^2)$  to the run time.
- Eisner's algorithm is the basis of current arc-factored dependency parsers.



## Some animations

- Presentation by Terry Koo and Michael Collins:
  - <http://people.csail.mit.edu/maestro/papers/koo10acl-talk.pdf>
- Also contains a bit more information than this lecture.



UPPSALA  
UNIVERSITET

# Evaluation of dependency parsing



# Evaluation of dependency parsers

- **labelled attachment score (LAS):**  
percentage of correct arcs,  
relative to the gold standard
- **labelled exact match (LEM):**  
percentage of correct dependency trees,  
relative to the gold standard
- **unlabelled attachment score/exact match (UAS/  
UEM):**  
the same, but ignoring arc labels



# Word- vs sentence-level AS

- **Example: 2 sentence corpus**  
sentence 1: 9/10 arcs correct  
sentence 2: 15/45 arcs correct
- **Word-level (*micro-average*):**  
 $(9+15)/(10+45) = 24/55 = 0.436$
- **Sentence-level (*macro-average*):**  
 $(9/10+15/45)/2 = (0.9+0.33)/2 = 0.617$
- Word-level attachment score is normally used





# Accuracy vs precision/recall

- Attachment score is an accuracy score
- For phrase-structure parsing we reported precision and recall
- Why is that not done for dependency parsing?



# Own work

- Coming lectures: Dec 12, Dec 14
- Start looking at the dependency assignments! (all)
- Start looking at the seminar article (all)
- Do assignment 2, literature review (m)
- Decide on a project and start on it! (m 7.5)
- Plan your workload for your two courses early, and remember to plan for holidays as well!