LLyfr cyntaf Moſes yr hwnaelwir GENESIS.

PENNOD I.

Creadwriaeth y nêf, a'r ddaiar, 2 Y goleuni a'r tywyllwch, 8 Yſſurfafen, 16 Y pyſc, yr adar, a'r anifeiliaid, 26 A dyn. 29 LLynniaeth dyn ac anifail.

YN y dechreuad y *creawdd Duw y nefoedd a'r ddaiar.

2 Y ddaiar oedd af- wyth eu rhywogaeth: a Duw a welodd mai da oedd.

13 Felly y bu hwyr a fu, a'r borau a fu, y trydydd dydd.

14 Duw hefyd a ddywedodd, *bydded goleuadau yn ſſurfafen y nefoedd i wahanu rhwng y dydd a'r nôs: a bydant yn arwyddion, ac yn dymmorau, ac yn ddyddiau, a bly-

Pſal.136.7. Deut.4.19.

*Heb.33.6. Pſal.136.5. Eccle.18.1. Act.14.15.

Dependency syntax

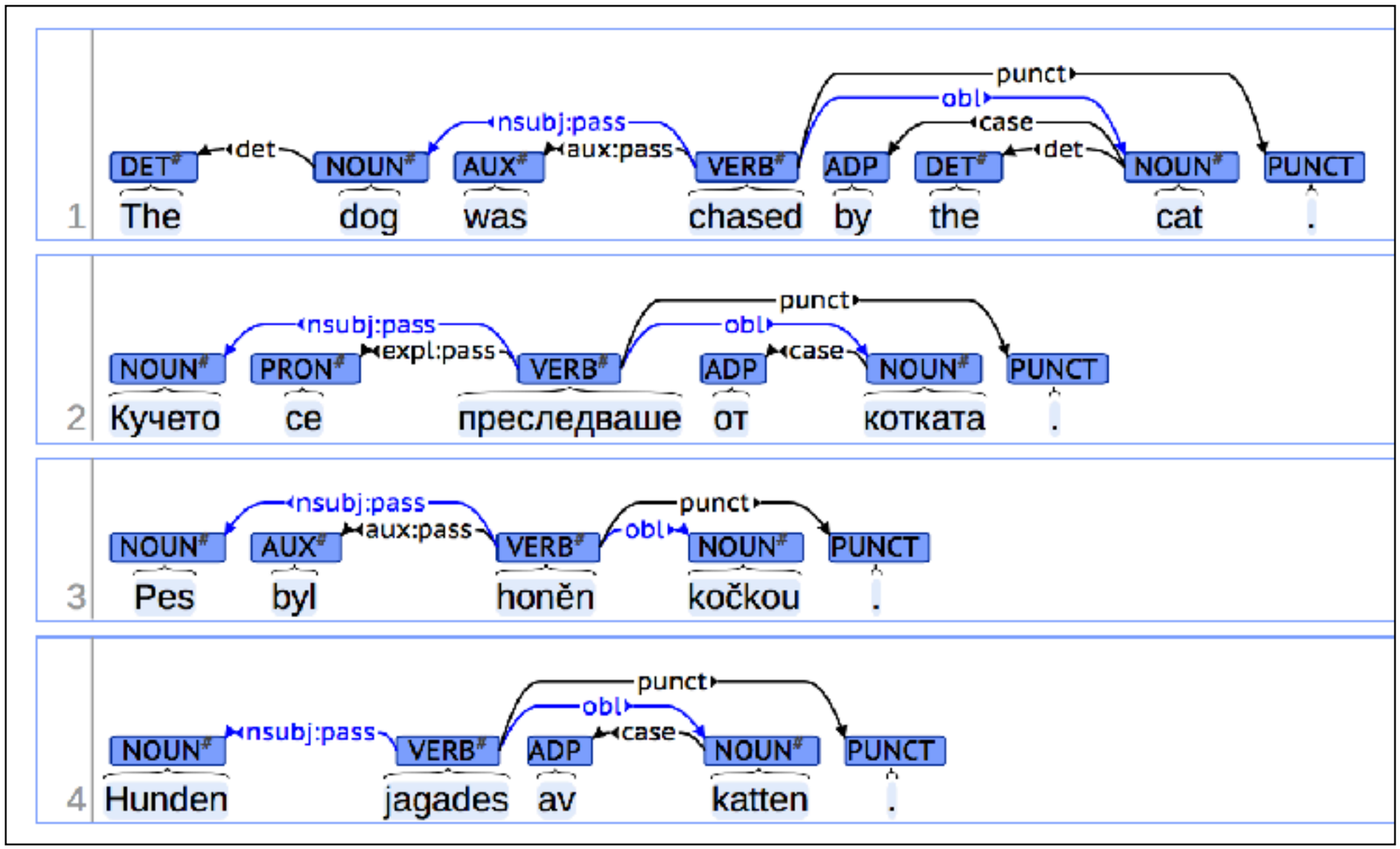
- Syntactic structure = asymmetric, binary relations between words.

Tesnier 1959; Nivre 2005

Trees

- A dependency structure is a directed graph G = (V,A) consisting of a set of vertices *V* and arcs *A* between them. Typically constrained to form a tree:
  - Single root vertex with no incoming arcs
  - Every vertex has exactly one incoming arc except root (single head constraint)
  - There is a unique path from the root to each vertex in V (acyclic constraint)

Universal Dependencies

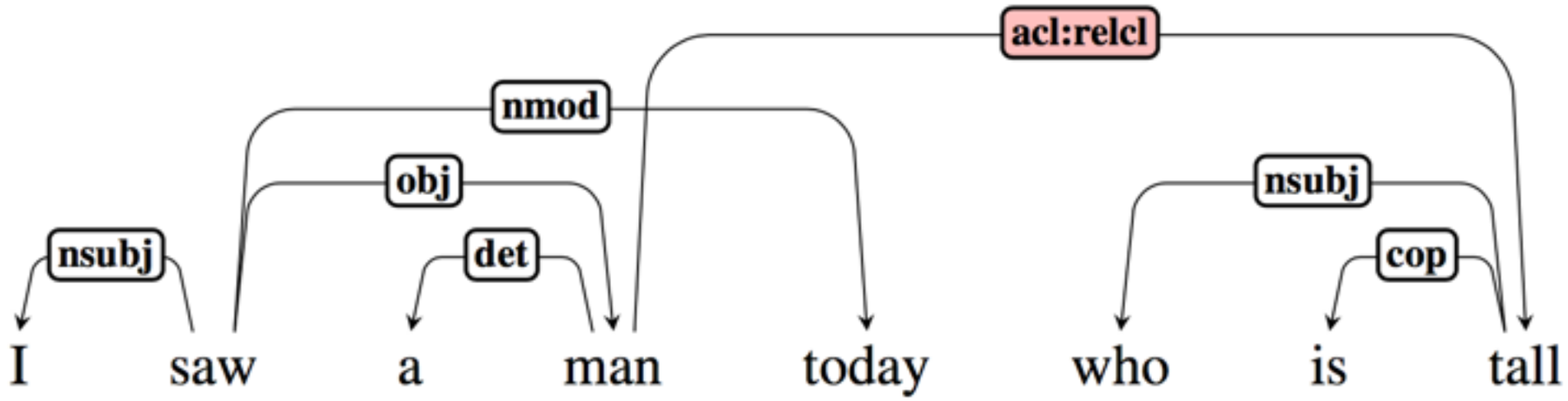


http://universaldependencies.org

Dependency parsing

- Transition-based parsing
  - O(n)
    - Only projective structures (pseudo-projective [Nivre and Nilsson 2005])
- Graph-based parsing
  - $O(n^2)$
  - Projective and non-projective trees

Projectivity



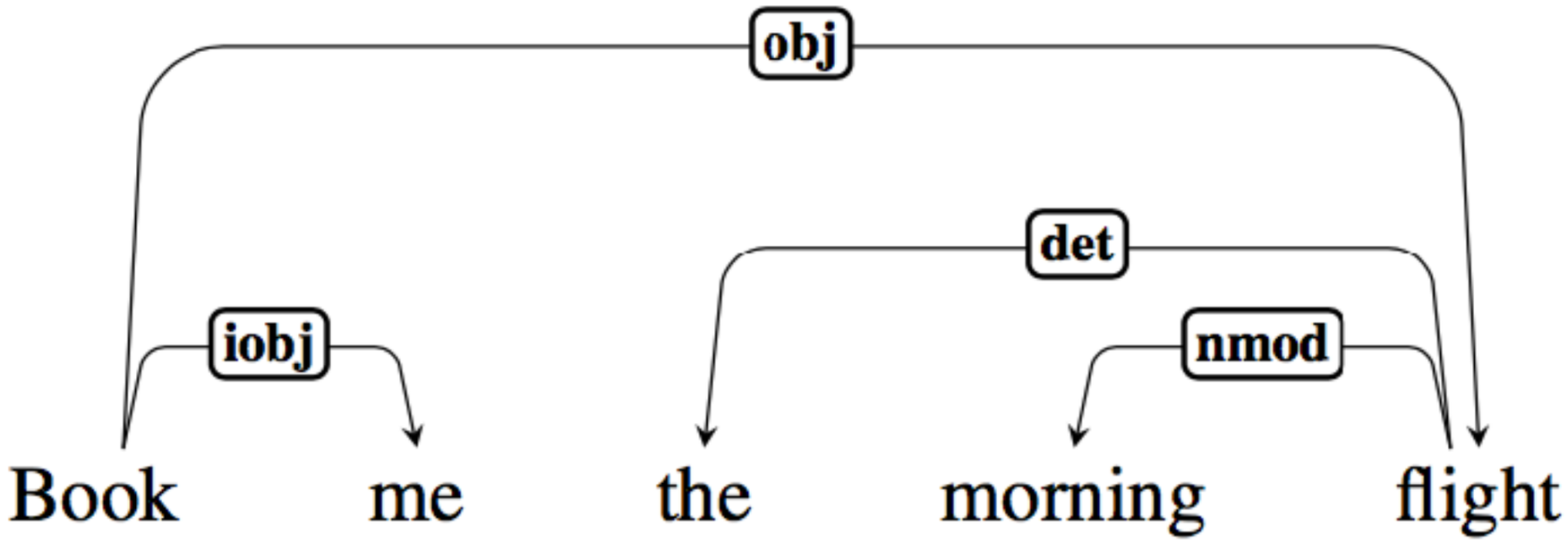
- An arc between a head and dependent is projective if there is a path from the head to every word between the head and dependent. Every word between head and dependent is a descendent of the head.

Transition-based parsing

- Basic idea: parse a sentence into a dependency by training a local classifier to predict a parser's next action from its current configuration.

Configuration

- Stack
- Input buffer of words
- Arcs in a parsed dependency tree
- Parsing = sequences of transitions through space of possible configurations


obj
det
iobj
nmod
Book
me
the
morning
flight

∅ book me the morning flight

stack

action

arc

∅ book me the morning flight

stack	action	arc

LeftArc(label): assert relation between head at $stack_1$ and dependent at $stack_2$: remove $stack_2$

RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$; remove $stack_1$

Shift: Remove word from front of input buffer (∅) and push it onto stack

∅ book me the morning flight

stack	action	arc

LeftArc(label): assert relation between head at $stack_1$ and dependent at $stack_2$: remove $stack_2$

RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$; remove $stack_1$

☞ Shift: Remove word from front of input buffer (∅) and push it onto stack

book me the morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ and dependent at $stack_2$: remove $stack_2$	
	RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$; remove $stack_1$	
∅	☞ Shift: Remove word from front of input buffer (∅) and push it onto stack	

book me the morning flight

stack	action	arc
∅	LeftArc(label): assert relation between head at $stack_1$ (∅) and dependent at $stack_2$: remove $stack_2$	
	RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$ (∅); remove $stack_1$ (∅)	
∅	Shift: Remove word from front of input buffer (book) and push it onto stack	

book me the morning flight

stack	action	arc
∅	☞	Shift: Remove word from front of input buffer (book) and push it onto stack
		LeftArc(label): assert relation between head at $stack_1$ (∅) and dependent at $stack_2$: remove $stack_2$
		RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$ (∅); remove $stack_1$ (∅)

me the morning flight

stack	action	arc
book	LeftArc(label): assert relation between head at $stack_1$ (∅) and dependent at $stack_2$: remove $stack_2$	
	RightArc(label): assert relation between head at $stack_2$ and dependent at $stack_1$ (∅); remove $stack_1$ (∅)	
∅	☞ Shift: Remove word from front of input buffer (book) and push it onto stack	

me the morning flight

stack	action	arc
book ∅	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅) RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book) Shift: Remove word from front of input buffer (me) and push it onto stack	

If we remove an element from the stack, it can't have any further dependents

me the morning flight

stack	action	arc
book	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
∅	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
	Shift: Remove word from front of input buffer (me) and push it onto stack	

If we remove an element from the stack, it can't have any further dependents

me the morning flight

stack	action	arc
book	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
∅	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
	☞ Shift: Remove word from front of input buffer (me) and push it onto stack	

If we remove an element from the stack, it can't have any further dependents

the morning flight

stack	action	arc
me	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	
book	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	
∅	☞ Shift: Remove word from front of input buffer (me) and push it onto stack	

the morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
me		
book	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
me	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
book	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
me	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
book		
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
		iobj(book, me)
book	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
∅	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
	Shift: Remove word from front of input buffer (the) and push it onto stack	

the morning flight

stack	action	arc
		iobj(book, me)
book	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
∅	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
	Shift: Remove word from front of input buffer (the) and push it onto stack	

morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
the	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
book	Shift: Remove word from front of input buffer (the) and push it onto stack	
∅		

morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (the) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
the book	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (the); remove $stack_1$ (the)	
∅	Shift: Remove word from front of input buffer (morning) and push it onto stack	

morning flight

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (the) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
the book	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (the); remove $stack_1$ (the)	
∅	☞ Shift: Remove word from front of input buffer (morning) and push it onto stack	

flight

stack	action	arc
morning the book ∅	LeftArc(label): assert relation between head at stack$_1$ (the) and dependent at stack$_2$ (book): remove stack$_2$ (book) RightArc(label): assert relation between head at stack$_2$ (book) and dependent at stack$_1$ (the); remove stack$_1$ (the) ☞ Shift: Remove word from front of input buffer (morning) and push it onto stack	*iobj(book, me)*

flight

stack	action	arc
.	LeftArc(label): assert relation between head at $stack_1$ (morning) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)*
morning		
the	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (morning); remove $stack_1$ (morning)	
book		
∅	Shift: Remove word from front of input buffer (flight) and push it onto stack	

flight

stack	action	arc
.		
morning	LeftArc(label): assert relation between head at $stack_1$ (morning) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)*
the	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (morning); remove $stack_1$ (morning)	
book		
∅	☞ Shift: Remove word from front of input buffer (flight) and push it onto stack	

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (morning) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)*
morning		
the	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (morning); remove $stack_1$ (morning)	
book		
∅	☞ Shift: Remove word from front of input buffer (flight) and push it onto stack	

stack	action	arc
flight . morning the book ∅	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (morning): remove $stack_2$ (morning) RightArc(label): assert relation between head at $stack_2$ (morning) and dependent at $stack_1$ (flight); remove $stack_1$ (flight) ~~Shift: Remove word from front of input buffer and push it onto stack~~	*iobj(book, me)* *nmod(flight, morning)*

stack		action	arc
flight	☞	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (morning): remove $stack_2$ (morning)	*iobj(book, me)*
.			*nmod(flight, morning)*
morning			
the		RightArc(label): assert relation between head at $stack_2$ (morning) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	
book			
∅		~~Shift: Remove word from front of input buffer and push it onto stack~~	

stack		action	arc
flight	☞	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (morning): remove $stack_2$ (morning)	*iobj(book, me)* *nmod(flight, morning)*
the		RightArc(label): assert relation between head at $stack_2$ (morning) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	
book		~~Shift: Remove word from front of input buffer and push it onto stack~~	
∅			

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)*
the	RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
book	~~Shift: Remove word from front of input buffer and push it onto stack~~	
∅		

stack		action	arc
flight	☞	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)* *nmod(flight, morning)*
the book ∅		RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (flight); remove $stack_1$ (flight) ~~Shift: Remove word from front of input buffer and push it onto stack~~	

stack		action	arc
flight	☞	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)* *nmod(flight, morning)*
the		RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*det(flight, the)*
book		~~Shift: Remove word from front of input buffer and push it onto stack~~	
∅			

stack		action	arc
flight	☞	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (the): remove $stack_2$ (the)	*iobj(book, me)* *nmod(flight, morning)*
book		RightArc(label): assert relation between head at $stack_2$ (the) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*det(flight, the)*
∅		~~Shift: Remove word from front of input buffer and push it onto stack~~	

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
book	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	*det(flight, the)*

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
book	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*nmod(flight, morning)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	*det(flight, the)*

stack	action	arc
flight	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
		nmod(flight, morning)
		det(flight, the)
book	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*obj(book, flight)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (flight) and dependent at $stack_2$ (book): remove $stack_2$ (book)	*iobj(book, me)*
		nmod(flight, morning)
		det(flight, the)
book	☞ RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (flight); remove $stack_1$ (flight)	*obj(book, flight)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	*iobj(book, me)*
		nmod(flight, morning)
book	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	*det(flight, the)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	*obj(book, flight)*

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	*iobj(book, me)*
		nmod(flight, morning)
		det(flight, the)
book	☞ RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	*obj(book, flight)*
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

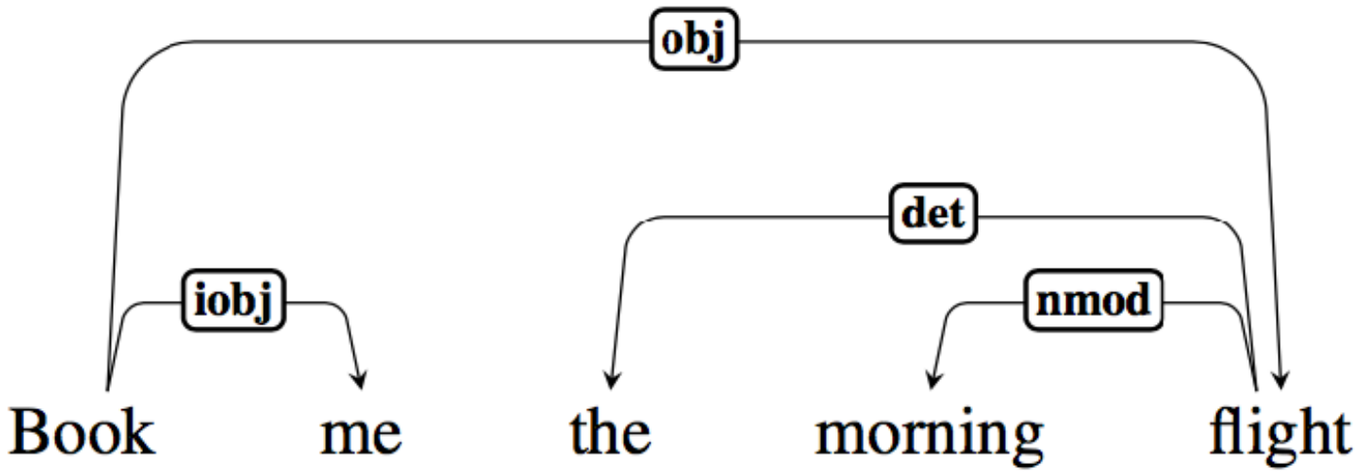
stack	action	arc
book ∅	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅) ☞ RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book) ~~Shift: Remove word from front of input buffer and push it onto stack~~	*iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	*iobj(book, me)*
		nmod(flight, morning)
		det(flight, the)
☞	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	*obj(book, flight)*
		root(∅, book)
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

This is our parse

stack	action	arc
	LeftArc(label): assert relation between head at $stack_1$ (book) and dependent at $stack_2$ (∅): remove $stack_2$ (∅)	*iobj(book, me)*
		nmod(flight, morning)
		det(flight, the)
☞	RightArc(label): assert relation between head at $stack_2$ (∅) and dependent at $stack_1$ (book); remove $stack_1$ (book)	*obj(book, flight)*
		root(∅, book)
∅	~~Shift: Remove word from front of input buffer and push it onto stack~~	

This is our parse



arc

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(∅, book)

Let's go back to this earlier configuration

the morning flight

stack	action	arc
me	LeftArc(label): assert relation between head at $stack_1$ (me) and dependent at $stack_2$ (book): remove $stack_2$ (book)	
book	RightArc(label): assert relation between head at $stack_2$ (book) and dependent at $stack_1$ (me); remove $stack_1$ (me)	
∅	Shift: Remove word from front of input buffer (the) and push it onto stack	

- This is a multi class classification problem: given the current configuration — i.e., the elements in the stack, the words in the buffer, and the arcs created so far, what's the best transition?

Output space $\boldsymbol{y}$ =

Shift
LeftArc(nsubj)
RightArc(nsubj)
LeftArc(det)
RightArc(det)
LeftArc(obj)
RightArc(obj)
...

stack

me

book

buffer

the morning flight

arc

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

Features are scoped over the stack, buffer, and arcs created so far

feature	example

stack

me

book

buffer

the morning flight

arc

Features are scoped over the stack, buffer, and arcs created so far

feature	example
$stack_1$ = me	1

stack

me

book

buffer

the morning flight

arc

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

feature	example
$stack_1$ = me	1
$stack_2$ = book	1

buffer

the morning flight

arc

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1

buffer

the morning flight

arc

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0
$buffer_1$ POS = RB	0

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0
$buffer_1$ POS = RB	0
$stack_1$ = me AND $stack_2$ = book	1

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0
$buffer_1$ POS = RB	0
$stack_1$ = me AND $stack_2$ = book	1
$stack_1$ = PRP AND $stack_2$ = VB	1

Features are scoped over the stack, buffer, and arcs created so far

stack

me

book

buffer

the morning flight

arc

feature	example
$stack_1$ = me	1
$stack_2$ = book	1
$stack_1$ POS = PRP	1
$buffer_1$ = the	1
$buffer_2$ = morning	1
$buffer_1$ = today	0
$buffer_1$ POS = RB	0
$stack_1$ = me AND $stack_2$ = book	1
$stack_1$ = PRP AND $stack_2$ = VB	1
iobj(book,*) in arcs	0

Use any multiclass classification model

- Logistic regression
- SVM
- NB
- Neural network

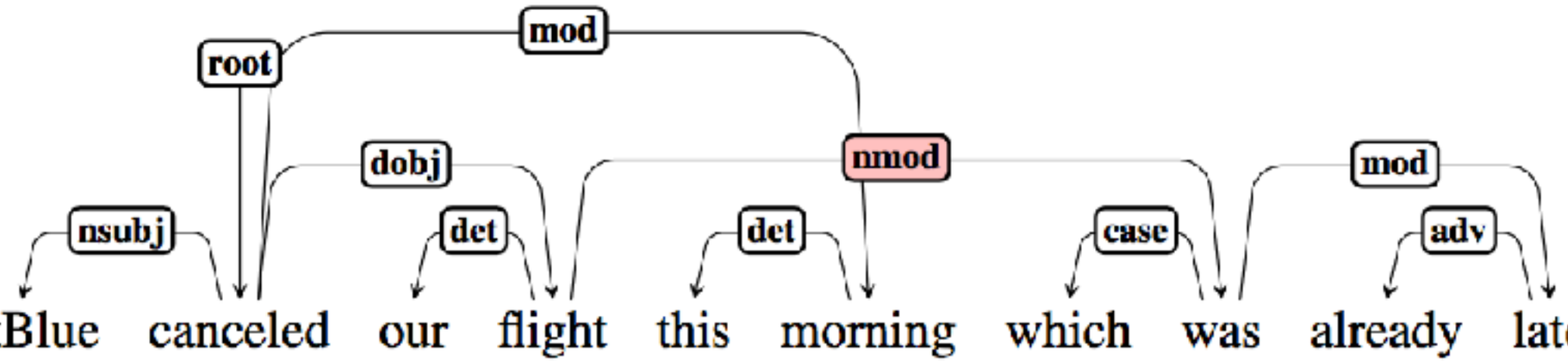
feature	example	β
$stack_1$ = me	1	0.7
$stack_2$ = book	1	1.3
$stack_1$ POS = PRP	1	6.4
$buffer_1$ = the	1	-1.3
$buffer_2$ = morning	1	-0.07
$buffer_1$ = today	0	0.52
$buffer_1$ POS = RB	0	-2.1
$stack_1$ = me AND $stack_2$ =	1	0
$stack_1$ = PRP AND $stack_2$ =	1	-0.1
iobj(book,*) in arcs	0	3.2

Training

We’re training to predict the parser action (Shift, RightArc, LeftArc) given the featurized configuration

Configuration features	Label
<stack1 = me, 1>, <stack2 = book, 1>, <stack1 POS = PRP, 1>, <buffer1 = the, 1>,	Shift
<stack1 = me, 0>, <stack2 = book, 0>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(det)
<stack1 = me, 0>, <stack2 = book, 1>, <stack1 POS = PRP, 0>, <buffer1 = the, 0>,	RightArc(nsubj)

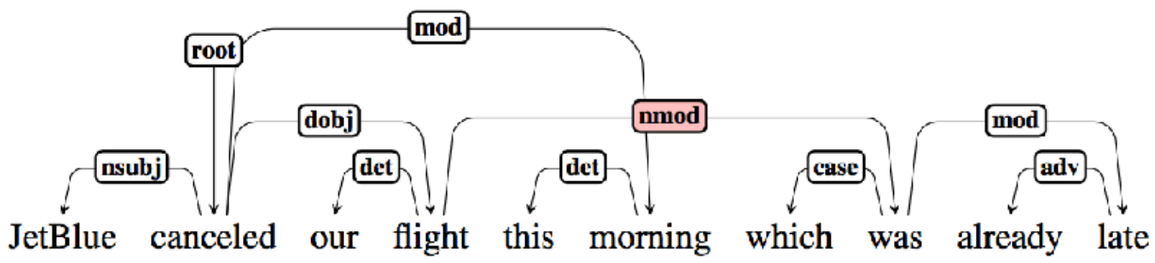
Training data



Our training data comes from treebanks (native dependency syntax or converted to dependency trees).

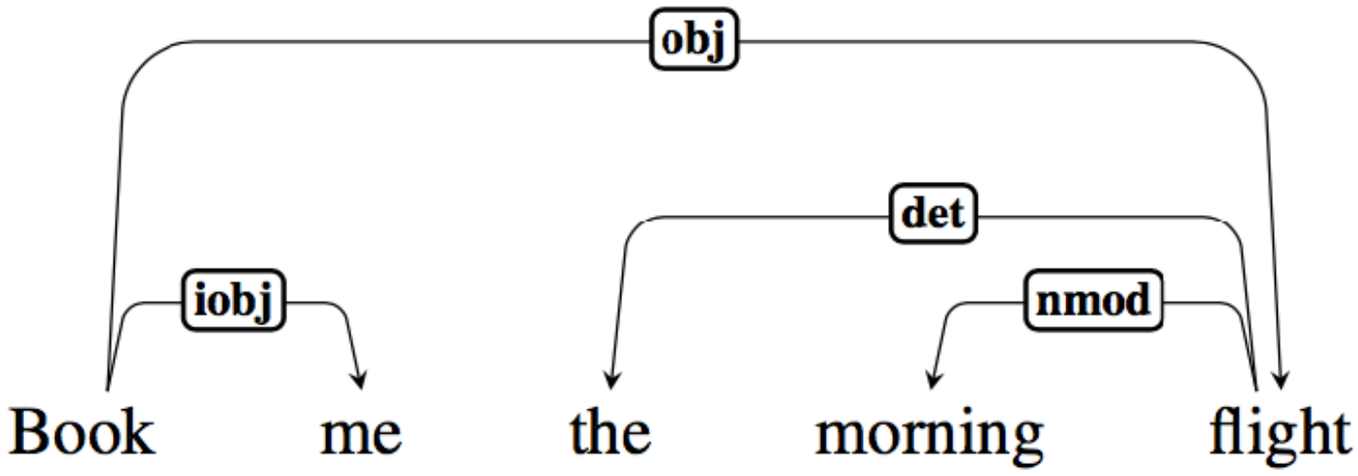
Oracle

- An algorithm for converting a gold-standard dependency tree into a series of actions a transition-based parser should follow to yield the tree.



Configuration	Label
<stack1 = me, 1>,	Shift
<stack1 = me, 0>,	RightArc(det)
<stack1 = me, 0>,	RightArc(nsu

This is our parse



arc

iobj(book, me)

nmod(flight, morning)

det(flight, the)

obj(book, flight)

root(∅, book)

∅ book me the morning flight

stack	action	gold tree
		iobj(book, me)
		nmod(flight, morning)
		det(flight, the)
		obj(book, flight)
		root(∅, book)

∅ book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
		det(flight, the)
		obj(book, flight)
	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
		det(flight, the)
		obj(book, flight)
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

book me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
		det(flight, the)
		obj(book, flight)
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
		nmod(flight, morning)
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

root(∅, book) exists but book has dependents in gold tree!

me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
		det(flight, the)
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

me the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
		nmod(flight, morning)
	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

the morning flight

stack	action	gold tree
me book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	*iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

iobj(book, me) exists and me has no dependents in gold tree

the morning flight

stack	action	gold tree
	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	*iobj(book, me)*
me	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*nmod(flight, morning)*
book	Else shift: Remove word from front of input buffer and push it onto stack	*det(flight, the)*
		obj(book, flight)
∅		*root(∅, book)*

the morning flight

stack	action	gold tree
me book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

the morning flight

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

morning flight

stack	action	gold tree
the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

morning flight

stack	action	gold tree
the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

stack	action	gold tree
morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

flight

flight

stack	action	gold tree
morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

stack	action	gold tree
flight morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

stack	action	gold tree
flight morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

nmod(flight,morning)

stack	action	gold tree
flight morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

nmod(flight,morning)

stack	action	gold tree
flight morning the book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* *det(flight, the)* *obj(book, flight)* *root(∅, book)*

nmod(flight,morning)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
the	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
the	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
the	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	*det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
the	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *det(flight, the)*
book		*obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

det(flight,the)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
book	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *nmod(flight, morning)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	✅ *det(flight, the)* *obj(book, flight)* *root(∅, book)*

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
book	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *det(flight, the)*
		obj(book, flight)
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

obj(book,flight)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
book	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *det(flight, the)*
		obj(book, flight)
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

obj(book,flight)

stack	action	gold tree
flight	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$.	✅ *iobj(book, me)*
		✅ *nmod(flight, morning)*
book	Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$	✅ *det(flight, the)*
		✅ *obj(book, flight)*
∅	Else shift: Remove word from front of input buffer and push it onto stack	*root(∅, book)*

obj(book,flight)

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* *root(∅, book)*

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* *root(∅, book)*

root(∅, book) *and* book has no more dependents we haven't seen

stack	action	gold tree
book ∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* *root(∅, book)*

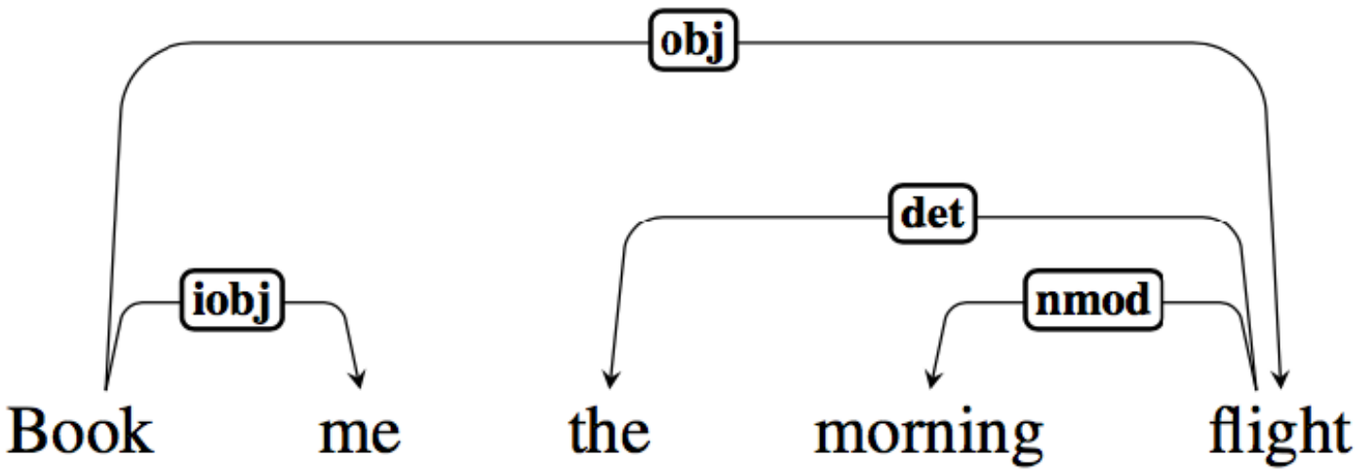
root(ø, book) *and* book has no more dependents we haven't seen

stack	action	gold tree
book ø	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* ✅ *root(ø, book)*

stack	action	gold tree
∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* ✅ *root(∅, book)*

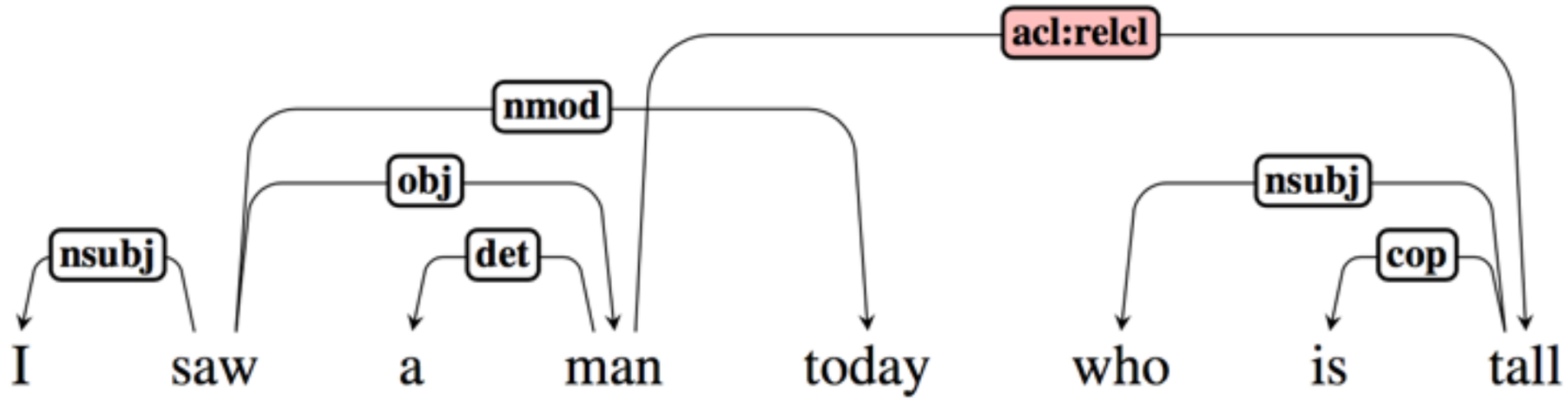
With only ∅ left on the stack and nothing in the buffer, we're done

stack	action	gold tree
∅	Choose LeftArc(label) if label($stack_1$,$stack_2$) exists in gold tree. Remove $stack_2$. Else choose RightArc(label) if label($stack_2$, $stack_1$) exists in gold tree and all arcs label($stack_1$, *). have been generated. Remove $stack_1$ Else shift: Remove word from front of input buffer and push it onto stack	✅ *iobj(book, me)* ✅ *nmod(flight, morning)* ✅ *det(flight, the)* ✅ *obj(book, flight)* ✅ *root(∅, book)*



Shift
Shift
Shift
RightArc(iobj)
Shift
Shift
Shift
LeftArc(nmod)
LeftArc(det)
RightArc(obj)
RightArc(root)

Projectivity



- What happens if you run an oracle on a non-projective sentence?

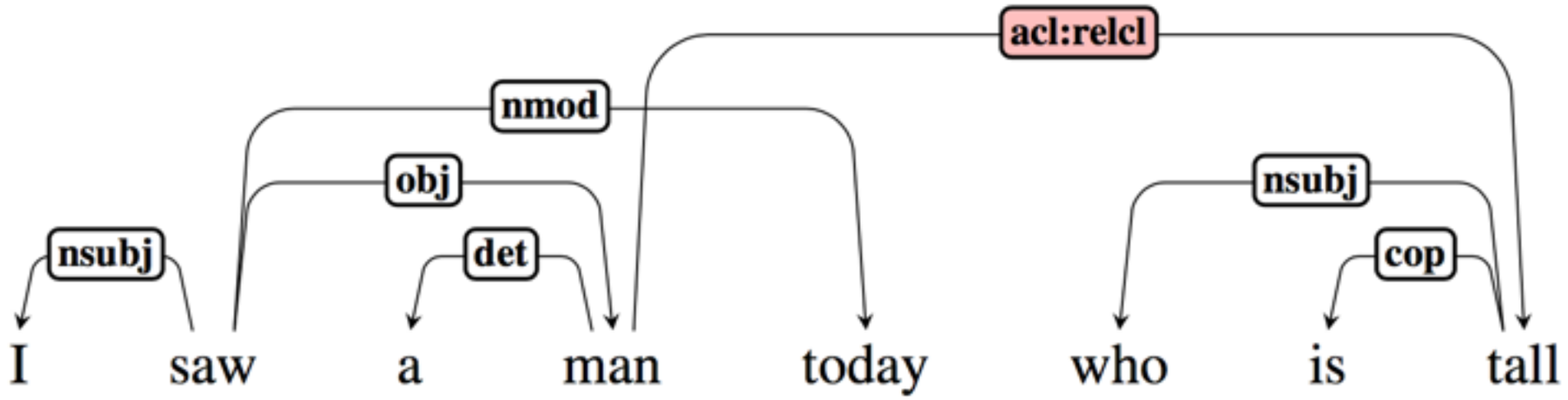
Graph-based parsing

- For a given sentence S, we want to find the highest-scoring tree among all possible trees for that sentence $\mathcal{G}_S$

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

- Edge-factored scoring: the total score of a tree is the sum of the scores for all of its edges (arcs):

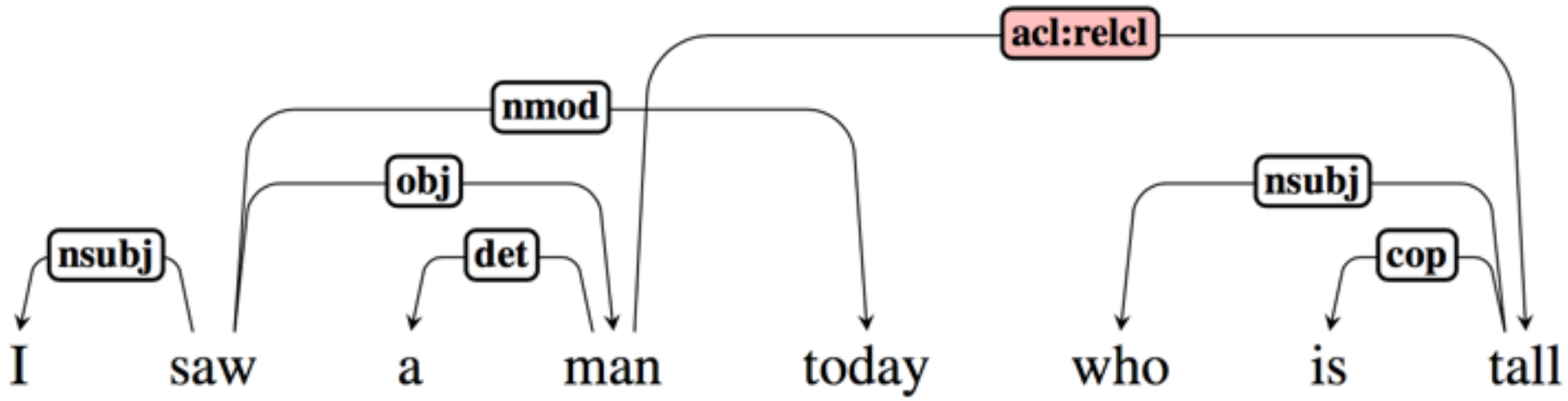
$$\text{score}(t, S) = \sum_{e \in t} \text{score}(e)$$



Edge-factored features

- Word form of head/dependent
- POS tag of head/dependent
- Distributed representation of h/d
- Distance between h/d
- POS tags between h/d
- Head to left of dependent?

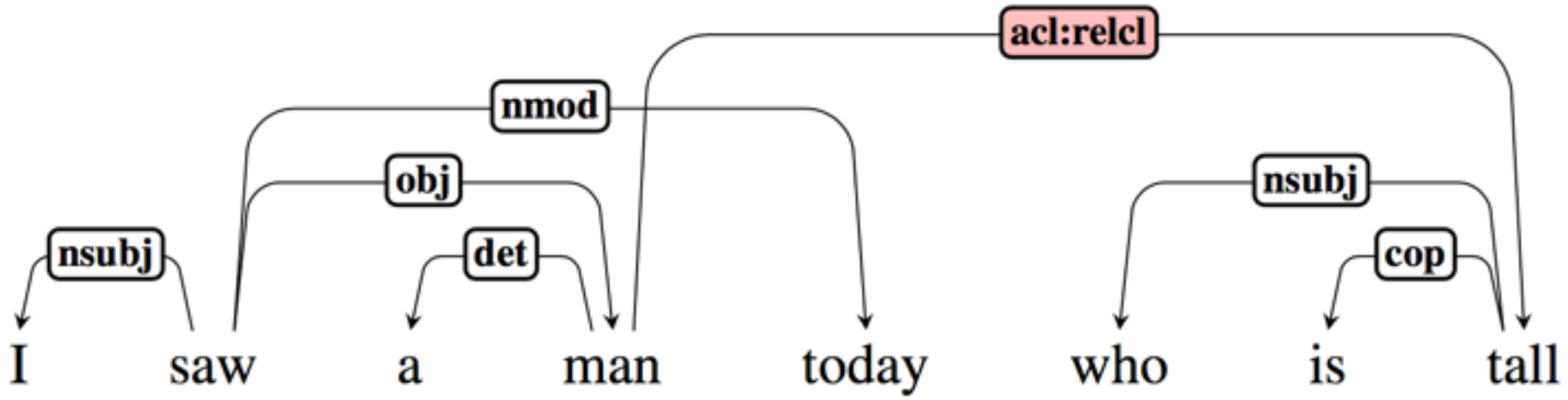
$head_t$ = man	1
$head_{pos}$ = NN	1
distance	4
$child_{pos}$ = JJ and $head_{pos}$ = NN	1
$child_{pos}$ = NN and $head_{pos}$ = JJ	0



$$\text{score}(e) = \sum_{i=1}^{F} x_i \beta_i$$

Feature value

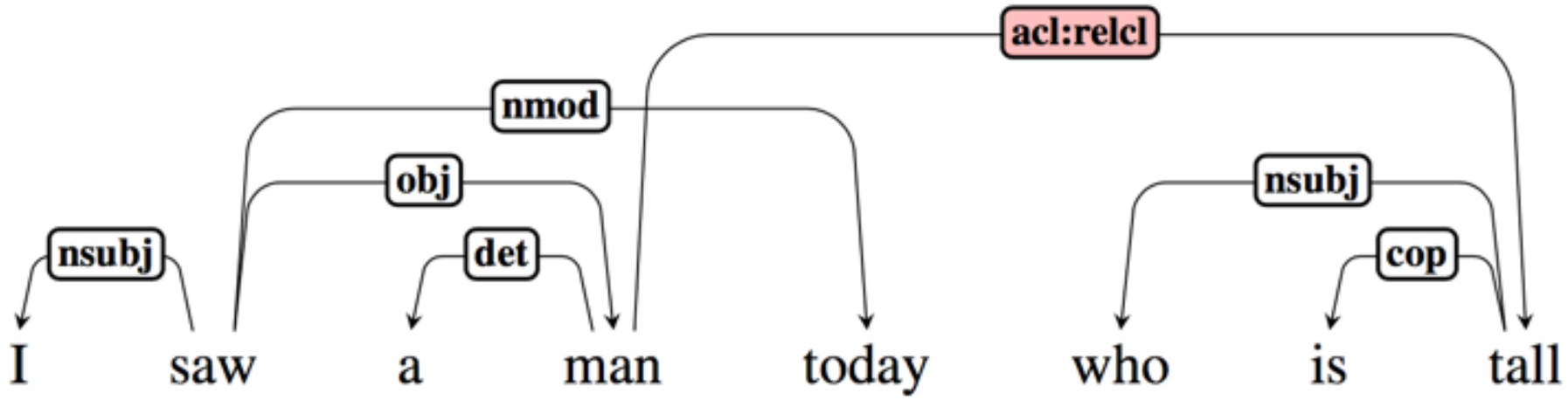
Learned coefficient for that feature

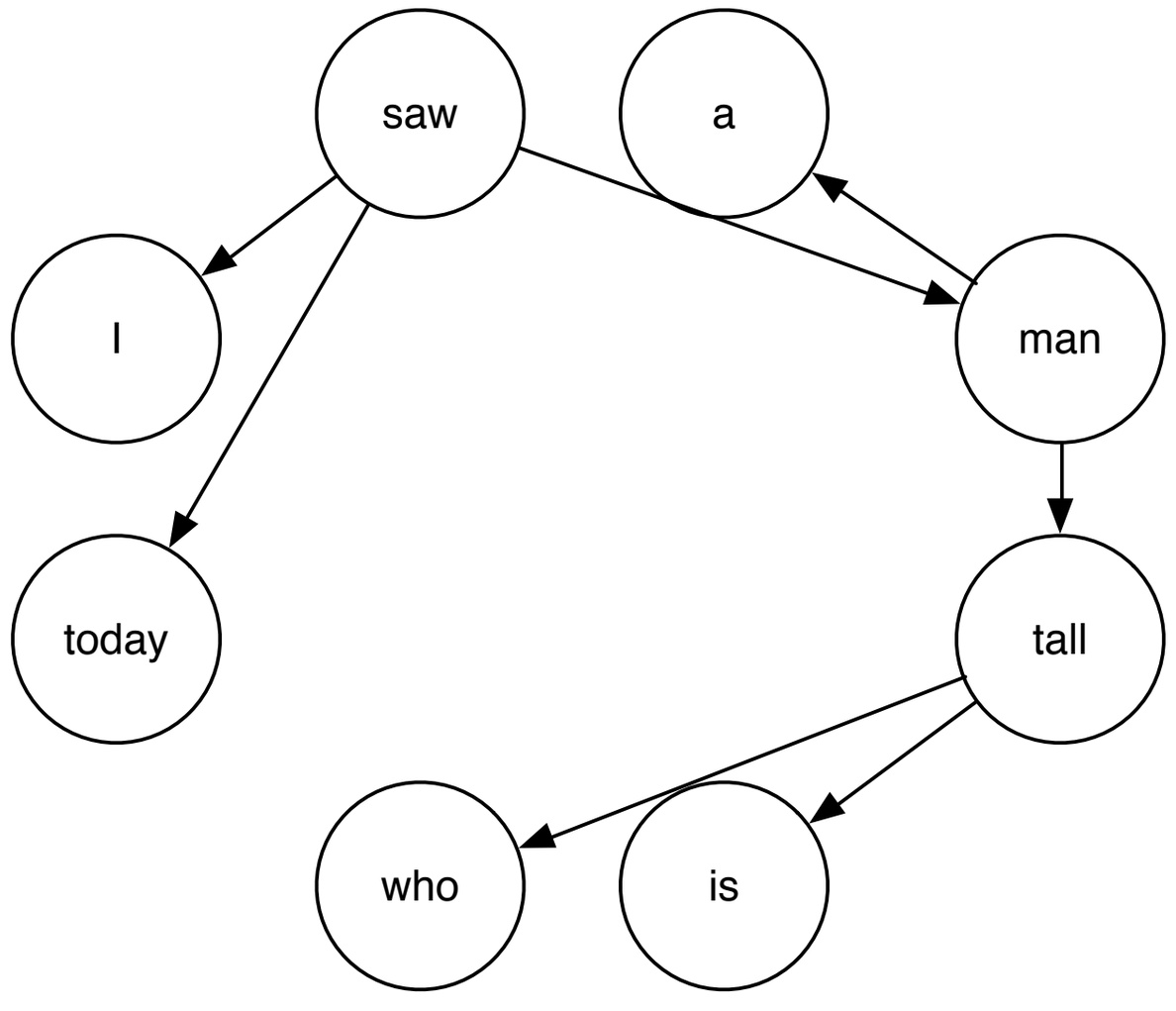


$$\text{score}(e) = \sum_{i=1}^{F} x_i \beta_i$$

	x	β
head_t = man	1	3.7
head_t = man	1	1.3
distance	4	0.7
child_{pos} = JJ and	1	0.3
child_{pos} = NN and	0	-2.7

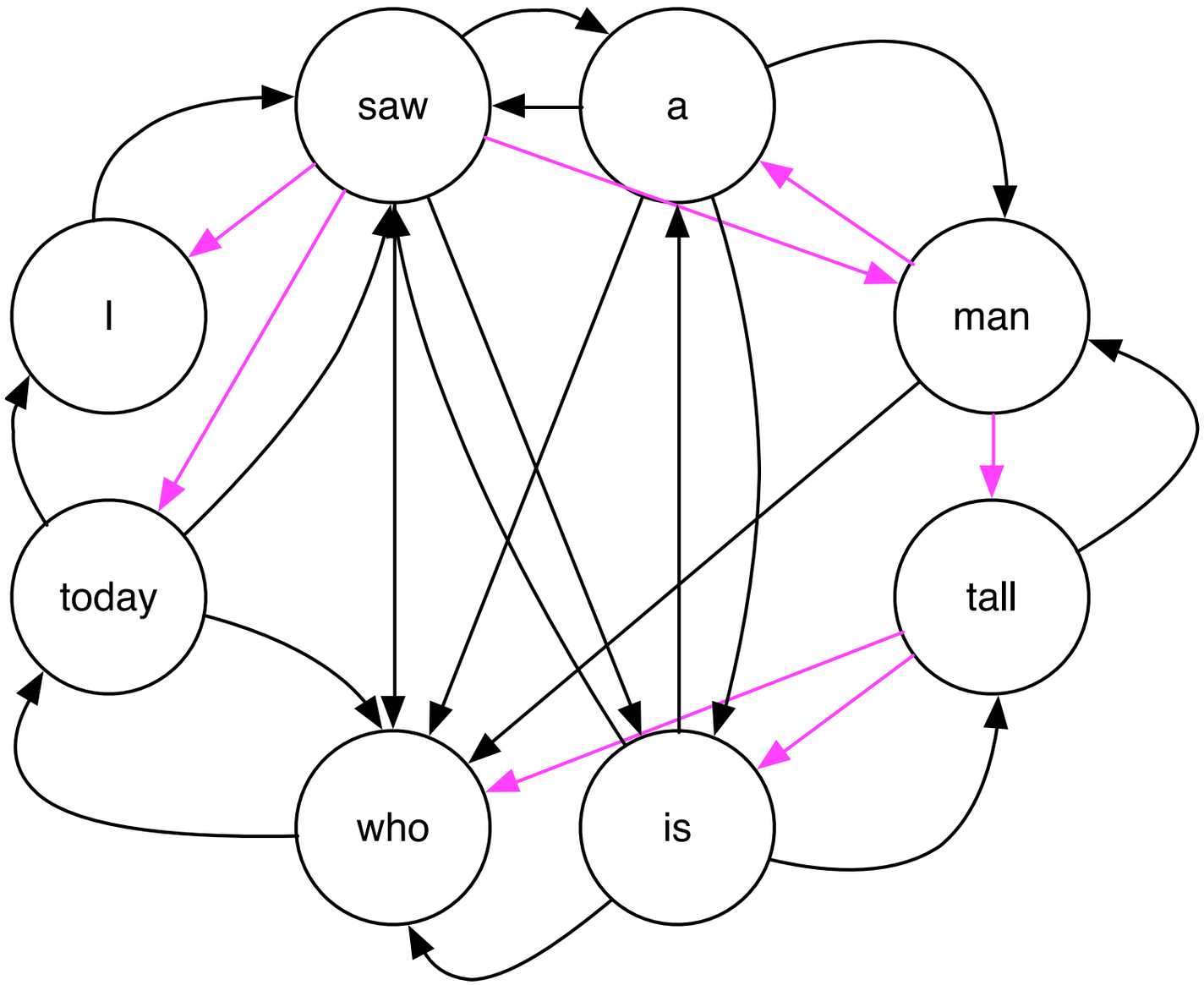
$$\text{score}(e) = 8.1$$


acl:relcl
nmod
obj
nsubj
det
nsubj
cop
I
saw
a
man
today
who
is
tall


saw
a
I
man
today
tall
who
is

MST Parsing

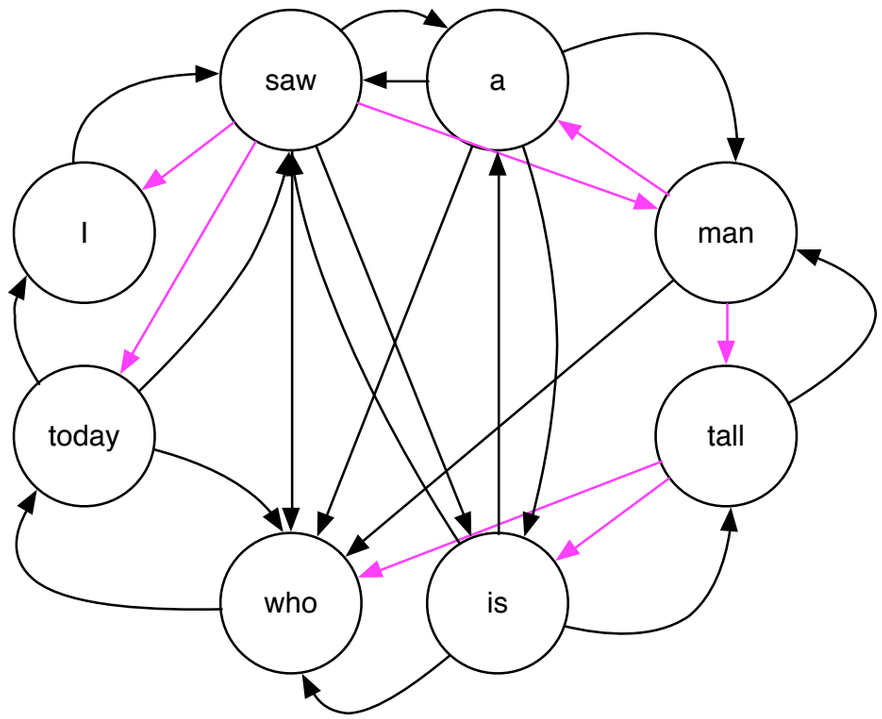
- We start out with a fully connected graph with a score for each edge
- N^2 edges total



(Assume one edge connects each node as dependent and node as head, N^2 total)

MST Parsing

- From this graph G, we want to find a spanning tree (tree that spans G [includes all the vertices in G])

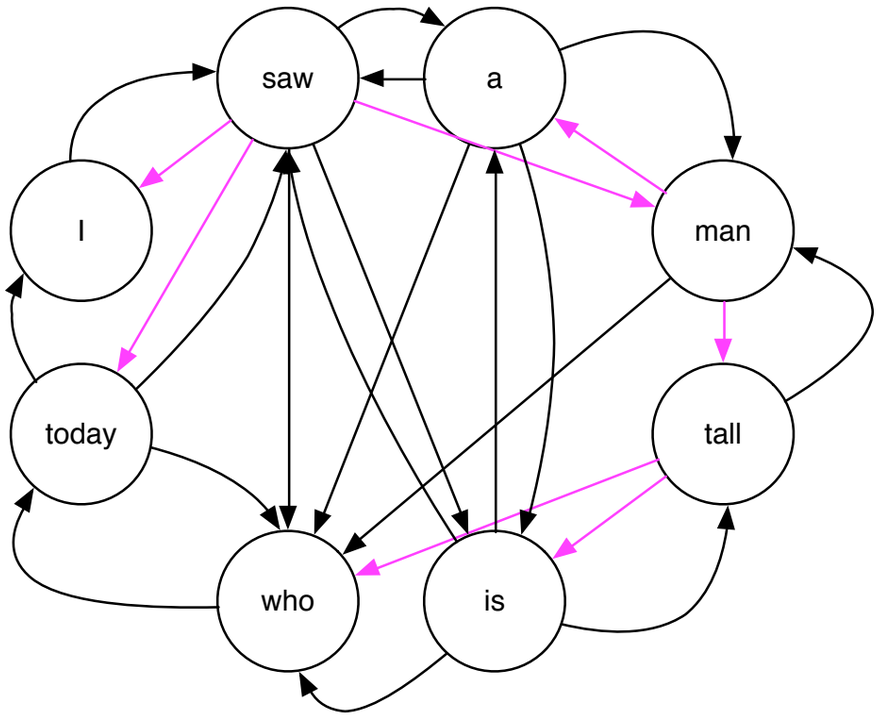


- If the edges have weights, the best parse is the maximal spanning tree (the spanning tree with the highest total weight).

MST Parsing

- To find the MST of any graph, we can use the Chu-Liu-Edmonds algorithm in $O(n^3)$ time.

- More efficient Gabow et al. find the MST in $O(n^2+n \log n)$



Learning

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

ϕ is our feature vector scoped over the source dependent, target head and entire sentence x

both are vectors

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \sum_{e \in E} \phi(e, x)^\top \beta$$

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \left[\sum_{e \in E} \phi(e, x) \right]^\top \beta$$

Learning

$$\hat{T}(S) = \arg\max_{t \in \mathcal{G}_S} \text{score}(t, S)$$

- Given this formulation, we want to learn weights for β that make the score for the gold tree higher than for all other possible trees.

- That's expensive, so let's just try to make the score for the gold tree higher than the single best tree we predict (if it's wrong)

Learning

$$\left[\sum_{e \in E} \phi(e, x)\right]^\top \beta = \Phi_{gold}(E, x)^\top \beta$$

score for gold tree in treebank

$$\left[\sum_{e \in \hat{E}} \phi(e, x)\right]^\top \beta = \hat{\Phi}_{gold}(\hat{E}, x)^\top \beta$$

score for argmax tree in our model

Learning

- We can optimize this using SGD by taking the derivative with respect to the difference in scores (which we want to maximize):

$$\Phi_{gold}(E, x)^\top \beta - \hat{\Phi}_{pred}(\hat{E}, x)^\top \beta$$

$$= \left[\Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)\right]^\top \beta$$

$$\frac{\partial}{\partial \beta}\left[\Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)\right]^\top \beta = \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$$

Perceptron

Data: training data $x \in R^F$, $y \in \{-1, +1\}$, i = 1 ... N;
initialize $\beta_0 = 0^F$;
k=0;
while *not converged* **do**
  k = k + 1;
  i = k mod N;
  if $\hat{y}_i \neq y_i$ **then**
    $\beta_{t+1} = \beta_t + y_i x_i$
  else
    do nothing;
  end
end

Perceptron

Data: training data $x \in R^F$, $y \in \{-1, +1\}$, i = 1 ... N;
initialize $\beta_0 = 0^F$;
k=0;
while *not converged* **do**
  k = k + 1;
  i = k mod N;
  if $\hat{y}_i \neq y_i$ **then**
    $\beta_{t+1} = \beta_t + y_i x_i$
  else
    do nothing;
  end
end

Perceptron update for binary classification = adding the feature values to the current estimate of β

Structured Perceptron

function PERCEPTRONUPDATE(x, E, β)
$\Phi_{gold}(E, x) \leftarrow$ createFeatures(x, E)
$\hat{E} \leftarrow \text{CLU}(x, \beta)$
$\hat{\Phi}_{pred}(\hat{E}, x) \leftarrow$ createFeatures(x, $\hat{E}$)
$\beta \leftarrow \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$
end function

Structured Perceptron

function PERCEPTRONUPDATE(x, E, β)
$\Phi_{gold}(E, x) \leftarrow$ createFeatures(x, E)
$\hat{E} \leftarrow \text{CLU}(x, \beta)$
$\hat{\Phi}_{pred}(\hat{E}, x) \leftarrow$ createFeatures(x, $\hat{E}$)
$\beta \leftarrow \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$
end function

Create feature vector from true tree

Structured Perceptron

function PERCEPTRONUPDATE(x, E, β)
$\Phi_{gold}(E, x) \leftarrow$ createFeatures(x, E)
$\hat{E} \leftarrow \text{CLU}(x, \beta)$
$\hat{\Phi}_{pred}(\hat{E}, x) \leftarrow$ createFeatures(x, $\hat{E}$)
$\beta \leftarrow \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$
end function

Create feature vector from true tree

Use CLU to find best tree given scores from current β

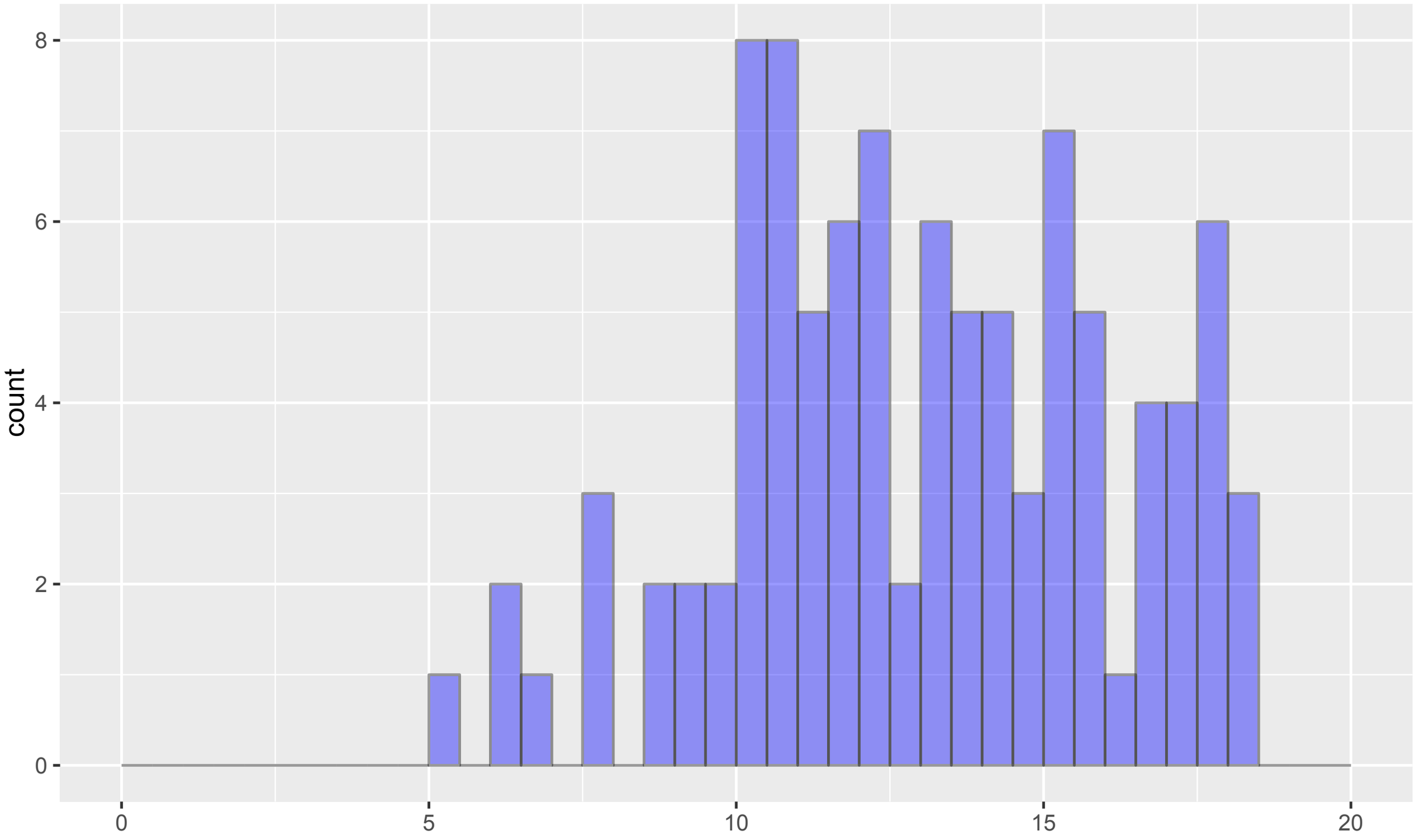
Structured Perceptron

function PERCEPTRONUPDATE(x, E, β)
$\Phi_{gold}(E, x) \leftarrow$ createFeatures(x, E)
$\hat{E} \leftarrow \text{CLU}(x, \beta)$
$\hat{\Phi}_{pred}(\hat{E}, x) \leftarrow$ createFeatures(x, $\hat{E}$)
$\beta \leftarrow \Phi_{gold}(E, x) - \hat{\Phi}_{pred}(\hat{E}, x)$
end function

Create feature vector from true tree

Use CLU to find best tree given scores from current β

Update β with the different between the feature vectors


count
8
6
4
2
0
0
5
10
15
20
Midterm scores

Thursday 10/26

- Guest lecture: Jacob Andreas

Announcements

- Midterm review (South Hall 210)
- Project midterm reports due 10/27
- DB no office hours Friday 10/27
- See TAs office hours for other midterm questions Friday