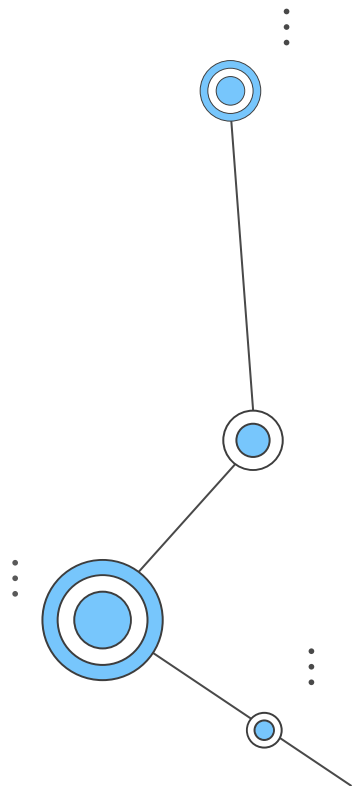


Transition-Based Dependency Parsing

2023/04/06

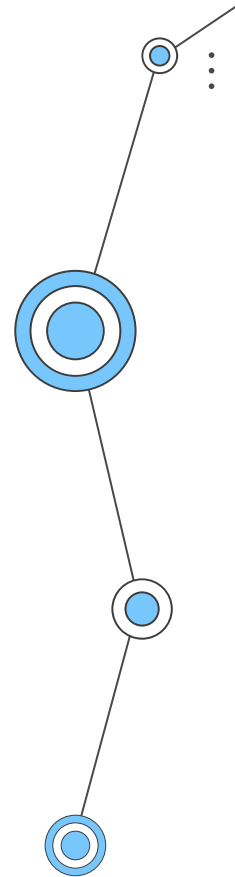


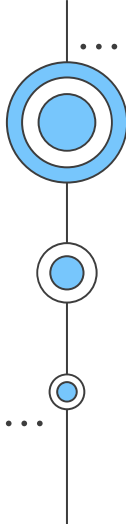
Tradition-Based

Arc Standard

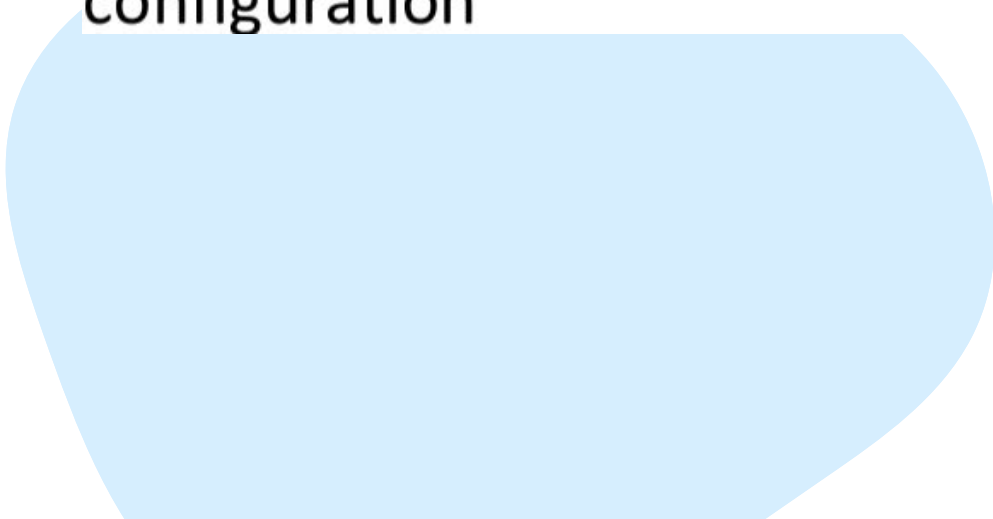
Arc Eager

Beam Search

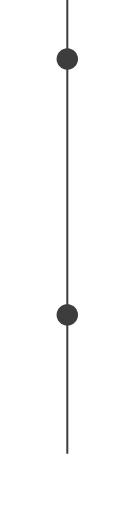
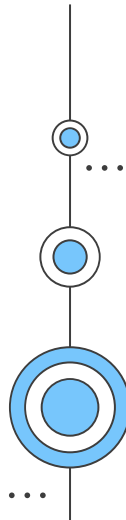




Transitions: produce a new configuration given current configuration



Parsing is the task of

- Finding a sequence of transitions
 - That leads from start state to desired goal state
- 
- 

Transitions: produce a new configuration given current configuration

- Start state

- Stack initialized with ROOT node
- Input buffer initialized with words in sentence
- Dependency relation set = empty



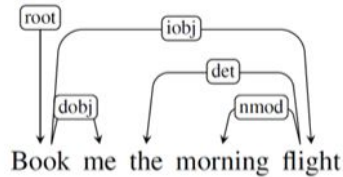
- End state

- Stack and word lists are empty
- Set of dependency relations = final parse

Parsing is the task of

- Finding a sequence of transitions
- That leads from start state to desired goal state

Transition-Based Parsing Illustrated



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 14.7 Trace of a transition-based parse.

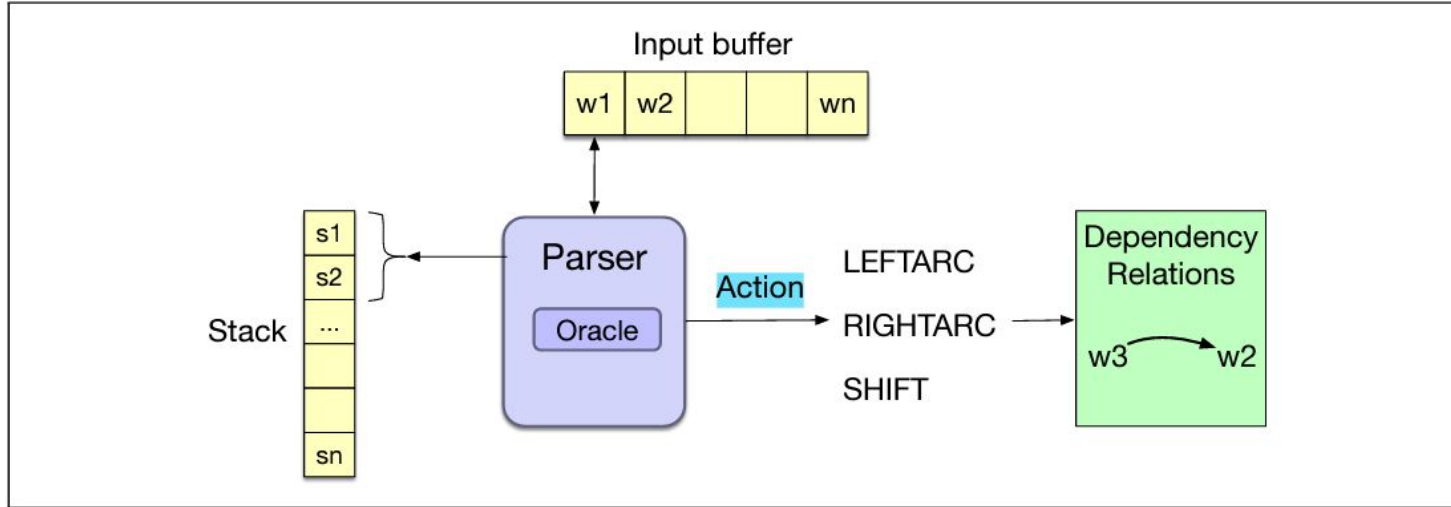
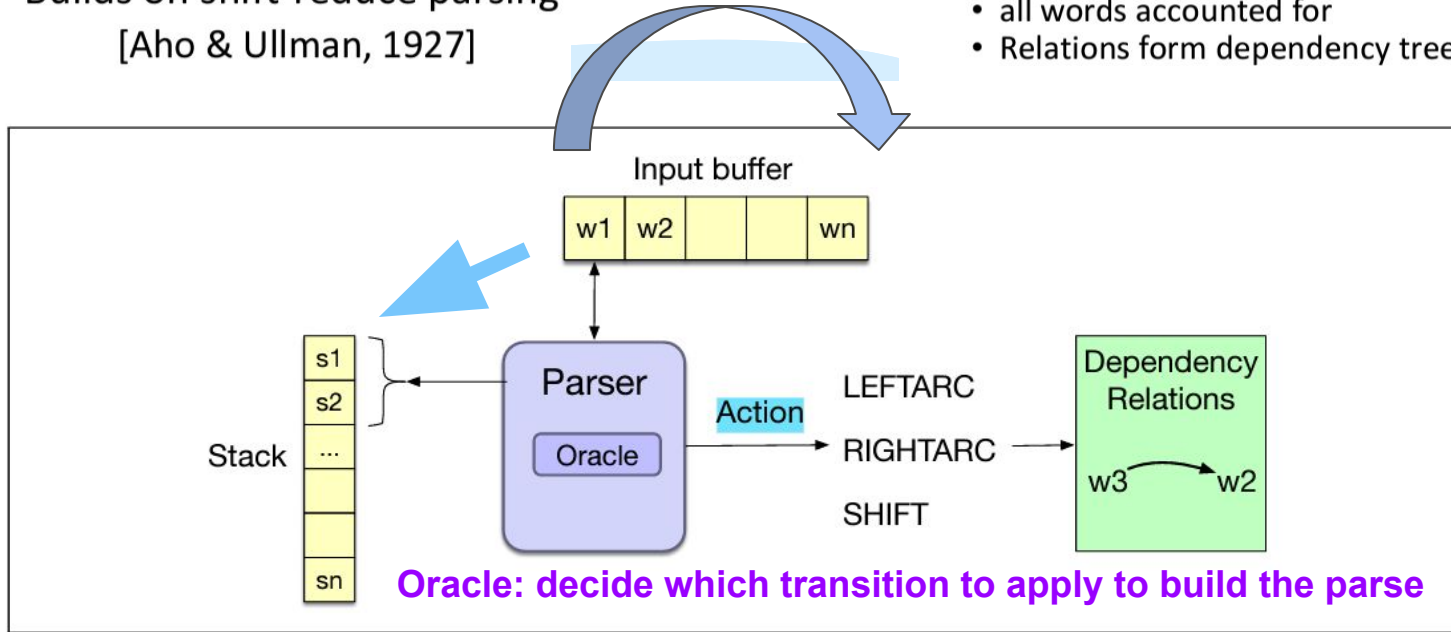


Figure 18.4 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action by consulting an oracle that examines the current configuration.

- Builds on shift-reduce parsing [Aho & Ullman, 1927]

- Goal of parsing
 - find a final configuration where
 - all words accounted for
 - Relations form dependency tree



Oracle: decide which transition to apply to build the parse

Figure 18.4 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action by consulting an oracle that examines the current configuration.



- **Transition Intuition**

- Assign the current word as the head of some previously seen word,
- Assign some previously seen word as the head of the current word,
- Postpone dealing with the current word, storing it for later processing.



**Reduce operations
(combine elements on the stack)**



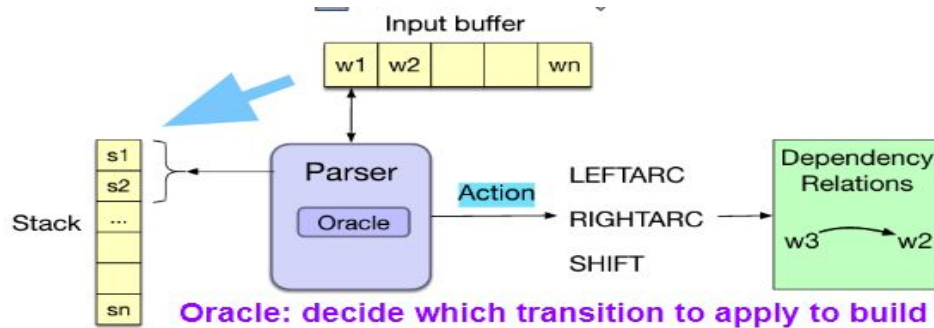
- **Transition Operators (Shift-Reduce Parsing)**

- **LEFTARC:** Assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack.
- **RIGHTARC:** Assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack;
- **SHIFT:** Remove the word from the front of the input buffer and push it onto the stack.



● Preconditions

- ROOT cannot have incoming arcs
- LEFT-ARC cannot be applied when ROOT is the 2nd element in stack
- LEFT-ARC and RIGHT-ARC require 2 elements in stack to be applied



● Oracle: decide which transition to apply to build the parse

- LEFTARC: Assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack.
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

Arc standard transition systems

The transition operators only assert relations between elements at the top of the stack, and once an element has been assigned its head it is removed from the stack and is not available for further processing.

**Reduce operations
(combine elements on the stack)**



- **Transition Operators (Shift-Reduce Parsing)**

- **LEFTARC:** Assert a head-dependent relation between the word at the top of the stack and the second word; remove the second word from the stack.
- **RIGHTARC:** Assert a head-dependent relation between the second word on the stack and the word at the top; remove the top word from the stack;
- **SHIFT:** Remove the word from the front of the input buffer and push it onto the stack.

- Parsing means making a sequence of transitions through the space of possible configurations.

- Start state

- Stack initialized with ROOT node
- Input buffer initialized with words in sentence
- Dependency relation set = empty



- End state

- Stack and word lists are empty
- Set of dependency relations = final parse

```
function DEPENDENCYPARSE(words) returns dependency tree
```

```
state ← { [root], [words], [] } ; initial configuration
```

```
while state not final
```

```
  t ← ORACLE(state)      ; choose a transition operator to apply
```

```
  state ← APPLY(t, state) ; apply it, creating a new state
```

```
return state
```

Figure 18.5 A generic transition-based dependency parser

- Builds on shift-reduce parsing [Aho & Ullman, 1927]

- Goal of parsing
 - find a final configuration where
 - all words accounted for
 - Relations form dependency tree

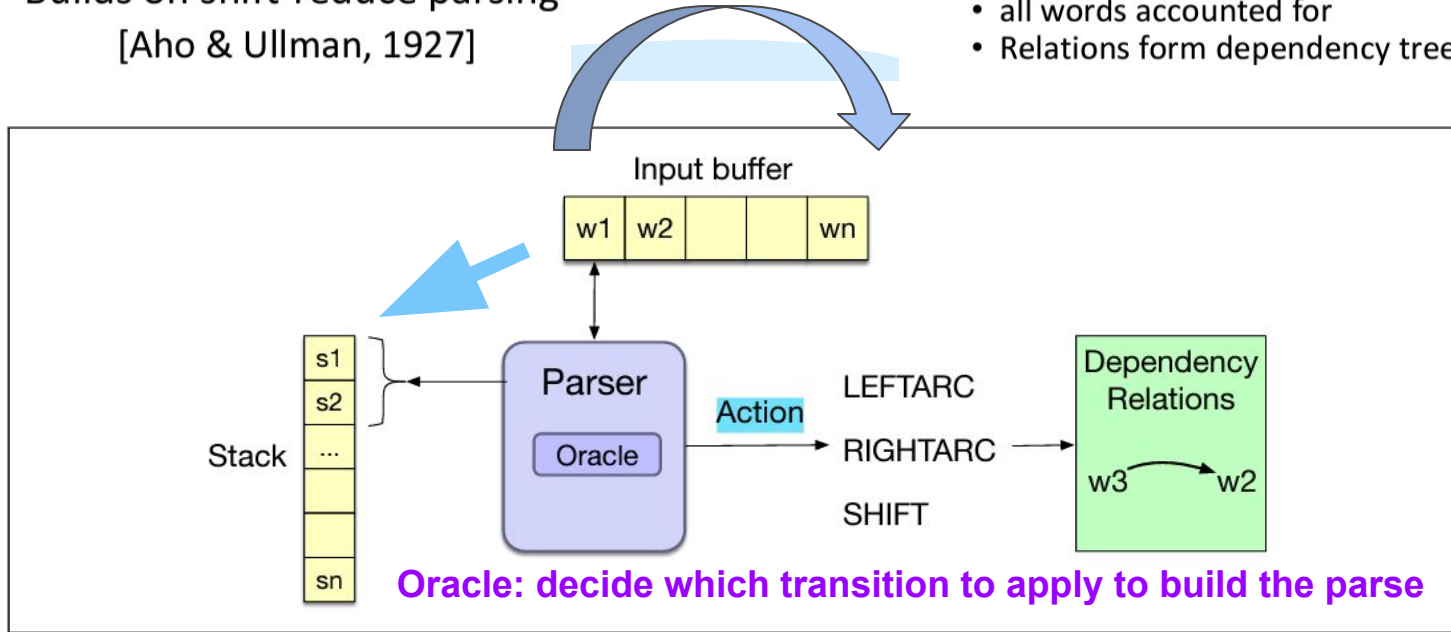
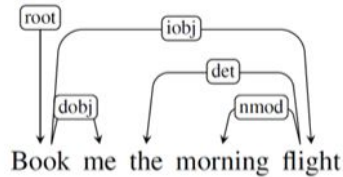


Figure 18.4 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action by consulting an oracle that examines the current configuration.

The process ends when all the words in the sentence have been consumed and the ROOT node is the only element remaining on the stack. Each word must first be shifted onto the stack and then later reduced.

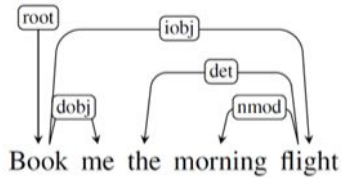
Transition-Based Parsing Illustrated



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 14.7 Trace of a transition-based parse.

Transition-Based

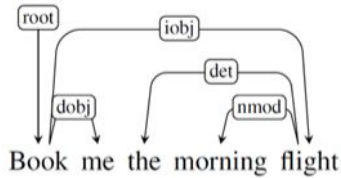


- LEFT-ARC:
 - create head-dependent rel. between word at top of stack and 2nd word (under top)
 - remove 2nd word from stack **NSUBJ**
- RIGHT-ARC:
 - Create head-dependent rel. between word on 2nd word on stack and word on top
 - Remove word at top of stack **OBJ**
- SHIFT
 - Remove word at head of input buffer
 - Push it on the stack

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 14.7 Trace of a transition-based parse.

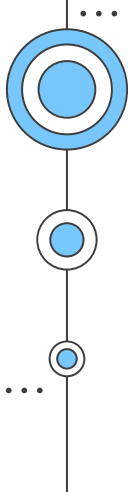
Transition-Based



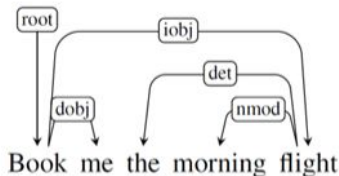
- LEFT-ARC:
 - create head-dependent rel. between word at top of stack and 2nd word (under top)
 - remove 2nd word from stack **NSUBJ**
- RIGHT-ARC:
 - Create head-dependent rel. between word on 2nd word on stack and word on top
 - Remove word at top of stack **OBJ**
- SHIFT
 - Remove word at head of input buffer
 - Push it on the stack

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)

Relation	Examples with <i>head</i> and dependent
NSUBJ	United <i>canceled</i> the flight.
OBJ	United <i>diverted</i> the flight to Reno. We <i>booked</i> her the first flight to Miami.



- LEFT-ARC:
 - create head-dependent rel. between word at top of stack and 2nd word (under top)
 - remove 2nd word from stack
- RIGHT-ARC:
 - Create head-dependent rel. between word on 2nd word on stack and word on top
 - Remove word at top of stack
- SHIFT
 - Remove word at head of input buffer
 - Push it on the stack



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	


At Step 1, LEFTARC is not applicable in the initial configuration since it asserts a relation, (root ← book), not in the reference answer; RIGHTARC does assert a relation contained in the final answer (root → book), however *book* has not been attached to any of its dependents yet, so we have to defer, leaving SHIFT as the only possible action. The same conditions hold in the next two steps. In step 3, LEFTARC is selected to link *the* to its head.





● Arc Eager Transition System

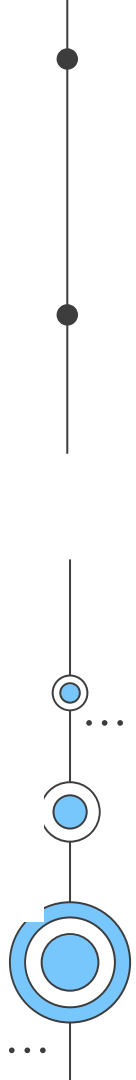
allowing words to be attached to their heads as early as possible, before all the subsequent words dependent on them have been seen.

- LEFTARC: Assert a head-dependent relation between the word at the front of the input buffer and the word at the top of the stack; pop the stack.
 - RIGHTARC: Assert a head-dependent relation between the word on the top of the stack and the word at the front of the input buffer; shift the word at the front of the input buffer to the stack.
 - SHIFT: Remove the word from the front of the input buffer and push it onto the stack.
 - REDUCE: Pop the stack.
- 



- **Arc Eager Transition System**

Differences

- The LEFTARC and RIGHTARC operators are applied to the **top of the stack** and **the front of the input buffer**
 - instead of **the top two elements of the stack** as in the **arc-standard approach**
 - The RIGHTARC operator now moves the dependent to the stack from the buffer rather than removing it
 - making it available to serve as the head of following words
 - The new REDUCE operator **removes the top element from the stack**
- 

Beam Search to Transition-based Parsing

- Apply all applicable operators to each state on an agenda and then score the resulting configurations
- Add each of these new configurations to the frontier, subject to the constraint that there has to be room within the beam
- Once the agenda reaches the limit, we only add new configurations that are better than the worst configuration on the agenda
- Assigning a score to all the possible transitions and picking the best one
- Define the score for a new configuration as the score of its predecessor plus the score of the operator used to produce it

$$\text{ConfigScore}(c_0) = 0.0$$

$$\text{ConfigScore}(c_i) = \text{ConfigScore}(c_{i-1}) + \text{Score}(t_i, c_{i-1})$$