Agenda

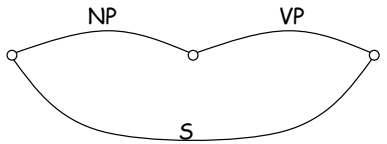
and

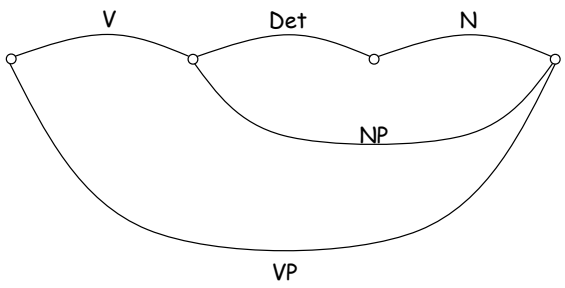
Chart

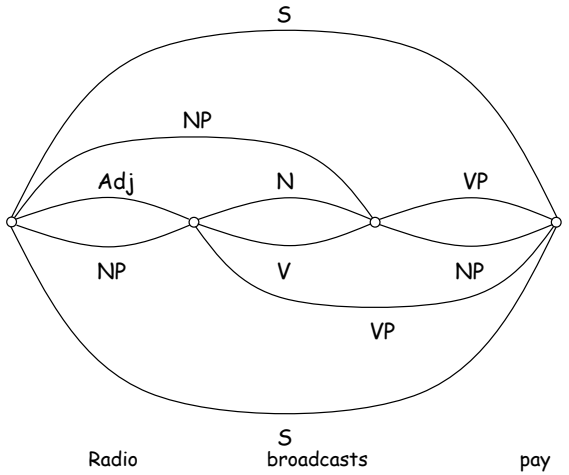
Martin Kay

Stanford University and
the University of the Saarland

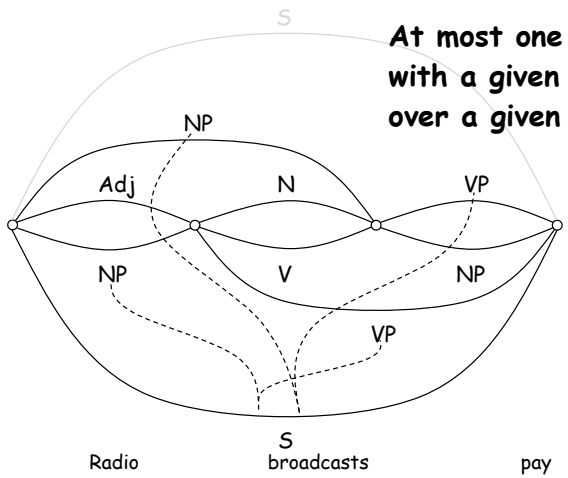
Charts



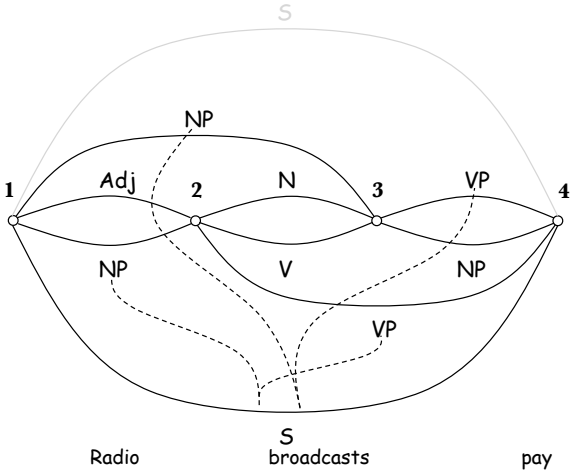


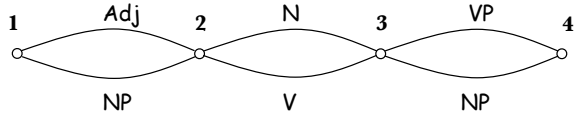


Structure



At most one edge with a given label over a given span





Adj	1	2
NP	1	2
N	2	3
V	2	3
NP	3	4
VP	3	4

Radio broadcasts pay

Adj	1	2
NP	1	2
N	2	3
V	2	3
NP	3	4
VP	3	4

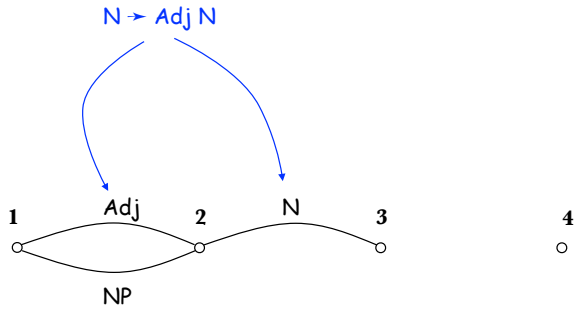
Radio broadcasts pay

NP	1	2
N	2	3
V	2	3
NP	3	4
VP	3	4

Radio broadcasts pay

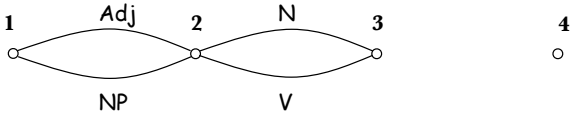
N	2	3
V	2	3
NP	3	4
VP	3	4

Radio broadcasts pay



V	2	3
NP	3	4
VP	3	4
NP	1	3

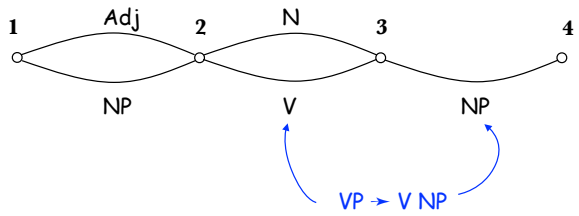
Radio broadcasts pay



NP	3	4
VP	3	4
NP	1	3

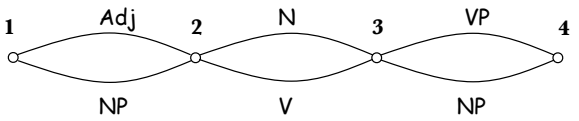
Radio broadcasts pay

VP	3	4
NP	1	3
VP	2	4



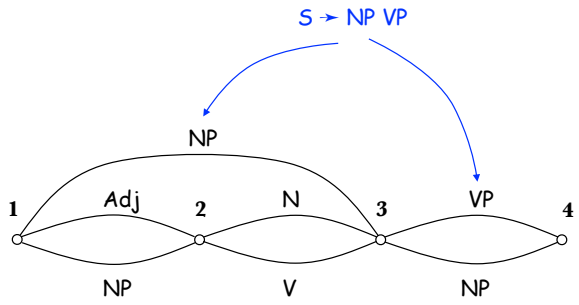
Radio broadcasts pay

NP	1	3
VP	2	4



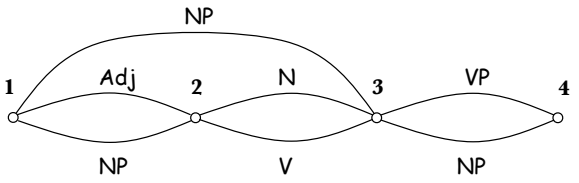
Radio broadcasts pay

VP	2	4



Radio broadcasts pay

VP	2	4
S	1	4



Radio broadcasts pay

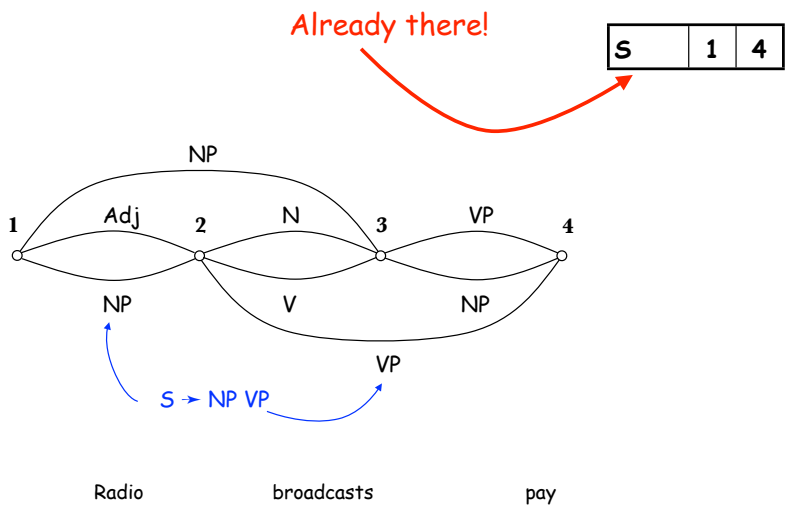
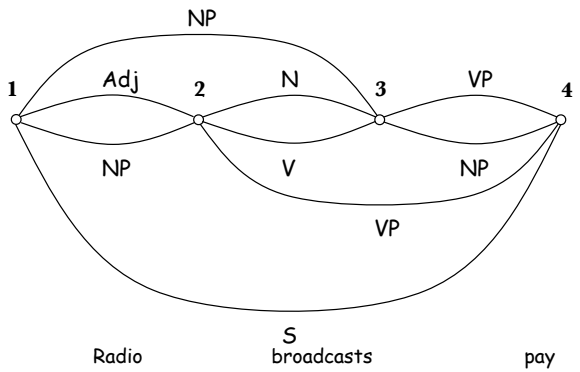


Chart-based, agenda-driven parsing without active edges.



Pseudo CKY

- **When a new edge is entered in the chart, apply**
  - —**apply all eligible rules and the edges to its left and right.**
  - —**apply unary rules**
- **This is pseudo-CKY because**
  - —**the order of events is not constrained to be length-first, left to right, etc.**
  - —**CKY does not provide for unary rules (though it requires only a small addition)**

Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Agenda

0-1	they
1-2	are
2-3	flying
3-4	planes

Chart

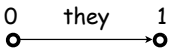
Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Agenda

1-2	are
2-3	flying
3-4	planes

Chart



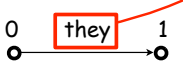
Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Agenda

1-2	are
2-3	flying
3-4	planes
0-1	NP

Chart



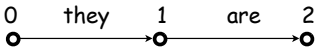
Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Agenda

2-3	flying
3-4	planes
0-1	NP

Chart



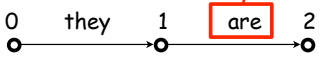
Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Agenda

2-3	flying
3-4	planes
0-1	NP
1-2	V
1-2	Be

Chart



Grammar

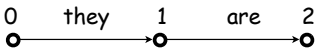
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

2-3 flying
3-4 planes
0-1 NP
1-2 V
1-2 Be

Chart



Grammar

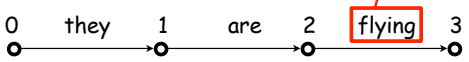
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

3-4 planes
0-1 NP
1-2 V
1-2 Be
2-3 Adj
2-3 PrPt

Chart



Grammar

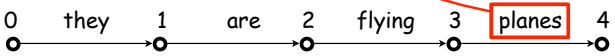
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-1 NP
1-2 V
1-2 Be
2-3 Adj
2-3 PrPt
3-4 N

Chart



Grammar

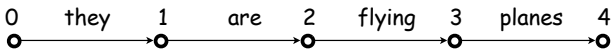
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-1 NP
1-2 V
1-2 Be
2-3 Adj
2-3 PrPt
3-4 N

Chart



Grammar

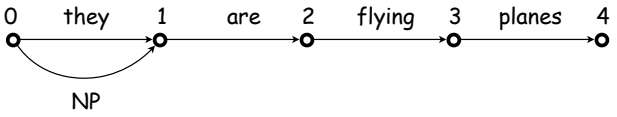
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-2 V
1-2 Be
2-3 Adj
2-3 PrPt
3-4 N

Chart



Grammar

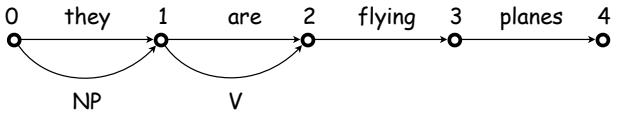
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-2 Be
2-3 Adj
2-3 PrPt
3-4 N

Chart



Grammar

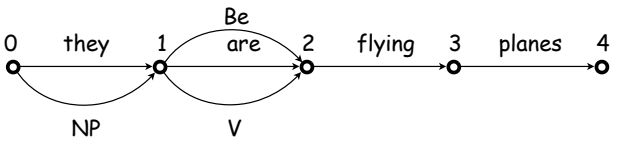
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

2-3 Adj
2-3 PrPt
3-4 N

Chart



No interactions

Grammar

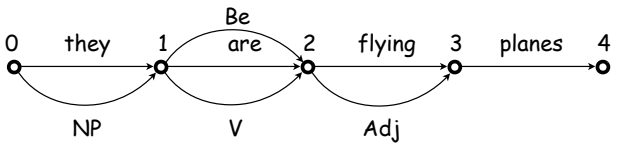
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

2-3 PrPt
3-4 N

Chart



No interactions

Grammar

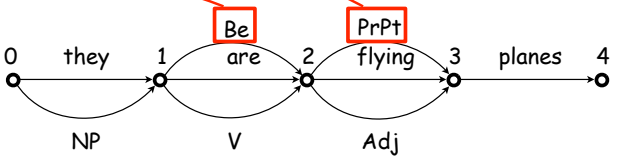
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

3-4 N
1-3 V

Chart



Grammar

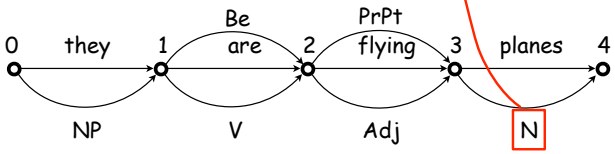
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-3 V
3-4 NP

Chart



Grammar

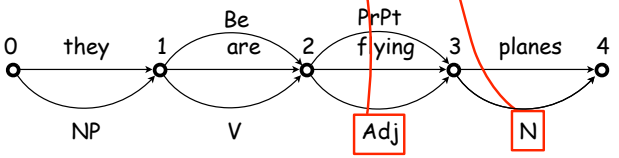
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-3 V
3-4 NP
2-4 N

Chart



Grammar

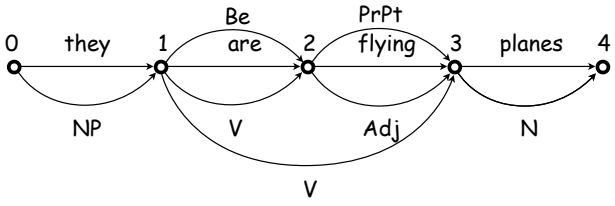
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

3-4 NP
2-4 N

Chart



Grammar

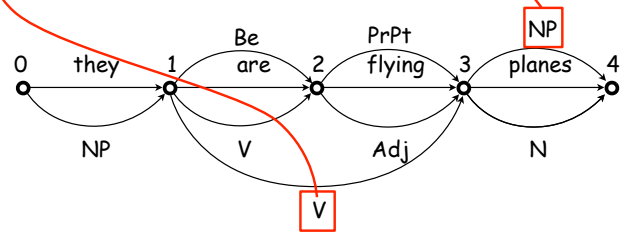
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

2-4 N
1-4 VP

Chart



Grammar

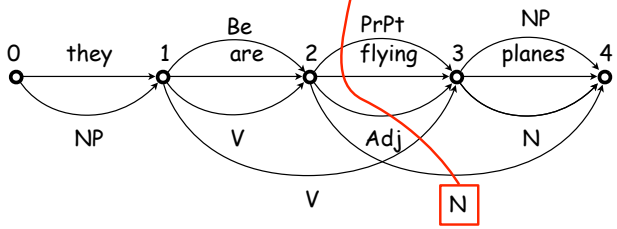
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-4 VP
2-4 NP

Chart



Grammar

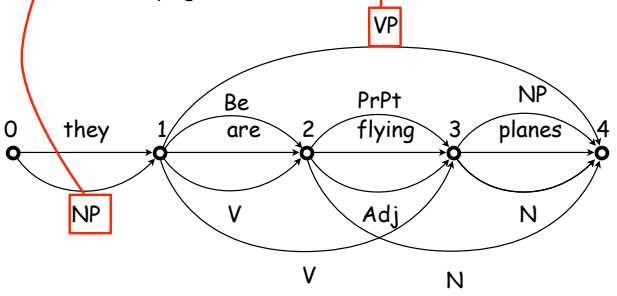
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

2-4 NP
0-4 S

Chart



Grammar

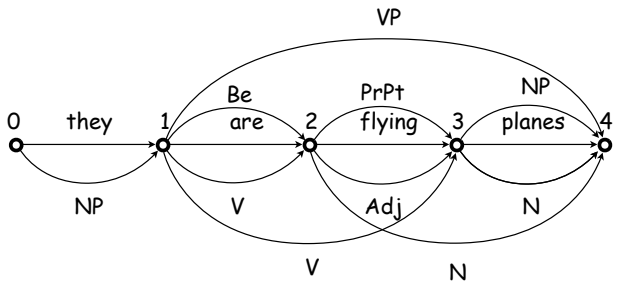
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

2-4 NP
0-4 S

Chart



Grammar

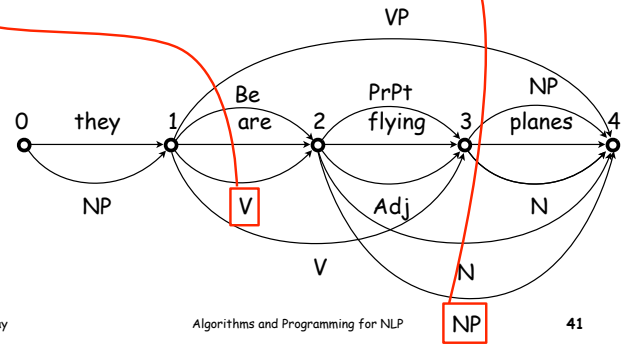
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-4 S
1-4 VP

Chart



Grammar

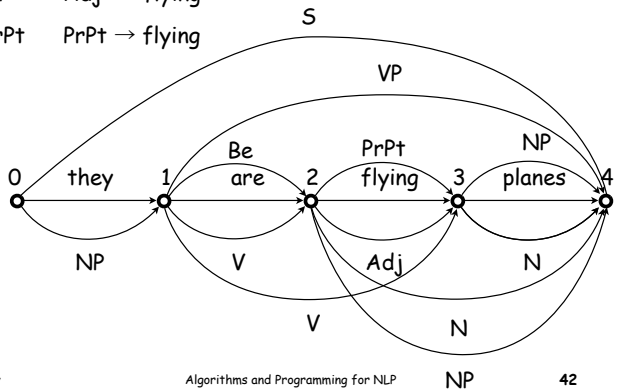
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-4 VP

Chart



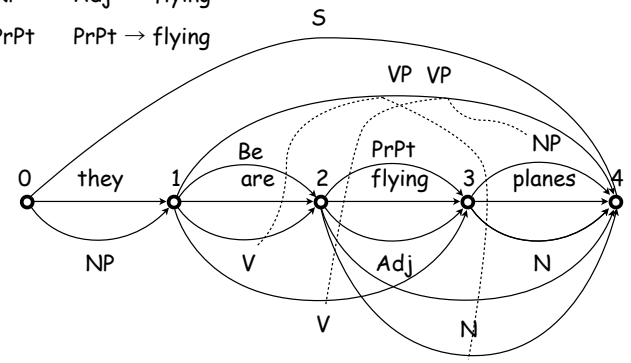
Grammar

S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

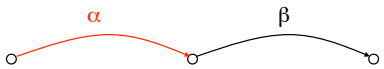
Agenda

Chart



Active Edges

The Chart Invariant—Active and Inactive



Every ∀α and β where

- —α is active and β is inactive, and
- —α is incident to, and β from, the same vertex, α is applied to β, and any resulting vertices added to the agenda

The Fundamental Rule

Grammar

S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt
NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Dotted Rules

Agenda

0-0 S → • NP VP
0-1 they
1-2 are
2-3 flying
3-4 planes

Chart

Grammar

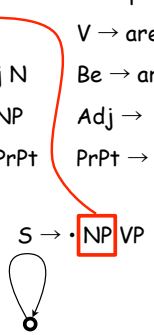
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt
NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-1 they
1-2 are
2-3 flying
3-4 planes

Chart

S → • NP VP



Grammar

S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt
NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-0 NP → • N
0-1 they
1-2 are
2-3 flying
3-4 planes

Chart

S → • NP VP

Grammar

S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-1 they
1-2 are
2-3 flying
3-4 planes

Chart

Grammar

S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

0-0 planes ...
0-1 they
1-2 are
2-3 flying
3-4 planes

Chart

Oops! Can't handle lexical items top-down

Grammar

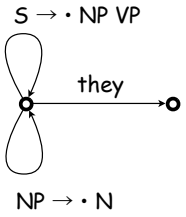
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-2 are
2-3 flying
3-4 planes

Chart



Grammar

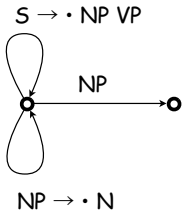
S → NP VP
NP → N
N → Adj N
VP → V NP
V → Be PrPt

NP → they
N → planes
V → are
Be → are
Adj → flying
PrPt → flying

Agenda

1-2 are
2-3 flying
3-4 planes

Chart



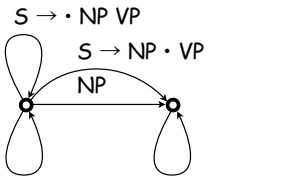
Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Agenda

1-2 are
2-3 flying
3-4 planes

Chart

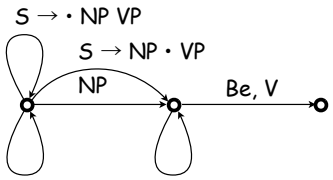


NP → • N VP → • V NP
V → • Be PrPt

Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Chart

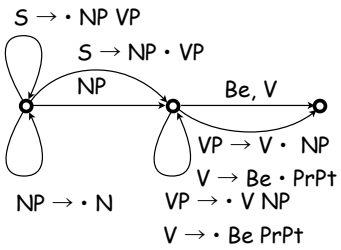


NP → • N VP → • V NP
V → Be PrPt

Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Chart

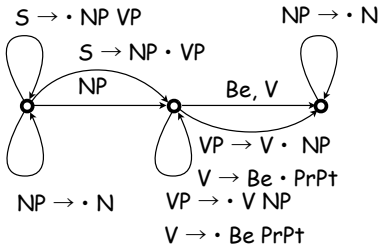


NP → • N VP → • V NP
V → • Be PrPt

Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Chart

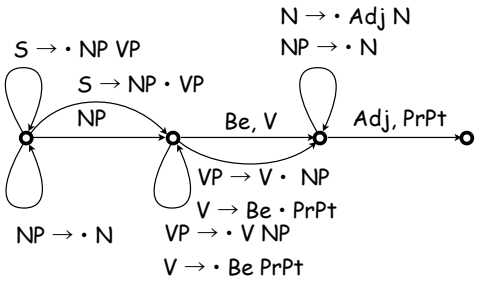


NP → • N VP → • V NP
V → • Be PrPt

Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

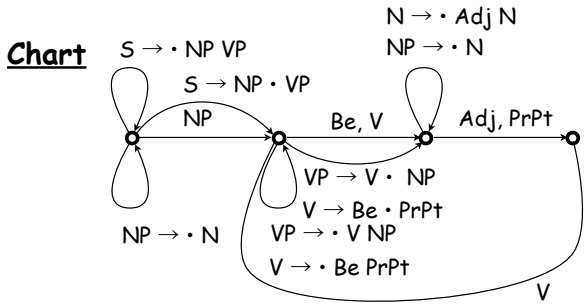
Chart



Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

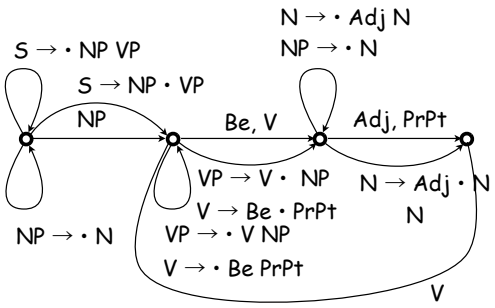
Chart



Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

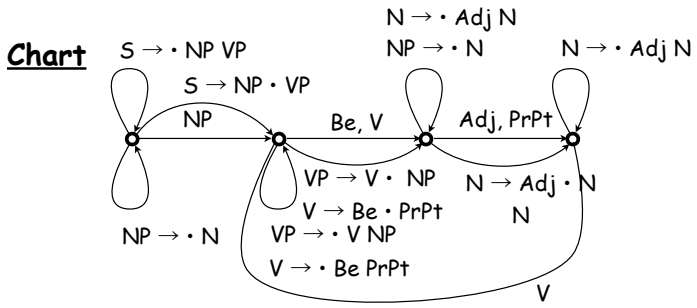
Chart



Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

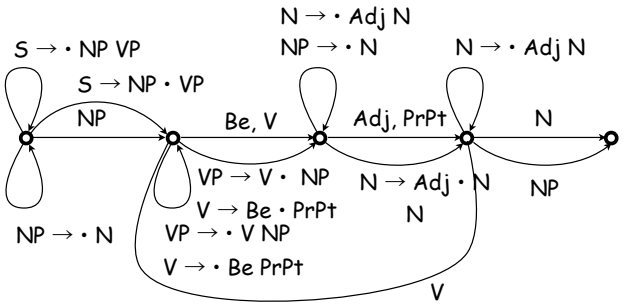
Chart



Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

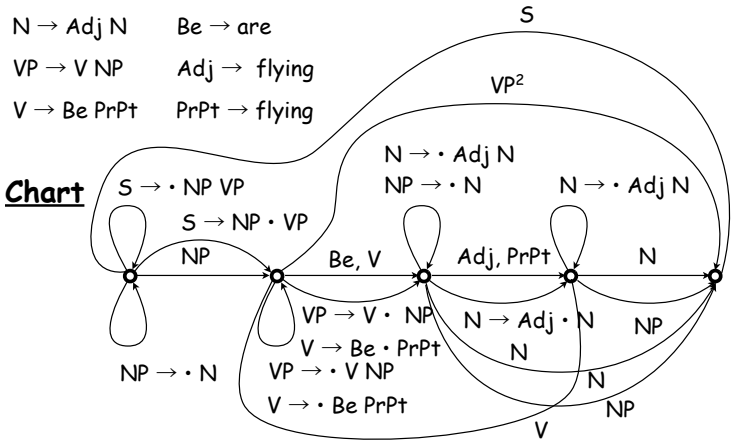
Chart



Grammar

S → NP VP	NP → they
NP → N	N → planes
N → Adj N	V → are
VP → V NP	Be → are
V → Be PrPt	Adj → flying
	PrPt → flying

Chart



Predictor

predicts more than necessary

Use reachability table to prune predictions?

Representation of Edges

- **Square array. Could use one side of diagonal for active and the other side for inactive edges—there are no inactive edges on the diagonal.**
- **Triangular array. For n-word sentence, store it in n(n+1)/2 consecutive locations. index(i, j) = i(i+1)/2+j.**

Representation of Edges

- List(s) of edges for each vertex.
- Active edges need to be accessed by their right-hand ends and inactive edges by their left hand ends. If active edges are not used, then edges need to accessed by both ends.

The agenda

- Duplicates have to be sought in both chart and agenda.
- Indexing by vertex is useful for this search, but is otherwise not required in the agenda.
- ∴ Put edges in chart immediately, but mark them as 'pending' until removed from the agenda. Advantage: search only one indexed data structure to check for duplicates.

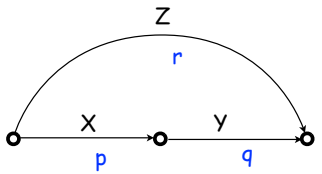
Stack Regime

Queue ⇒ Breadth first search

Stack ⇒ Depth first search

Priority sort ⇒ Best first

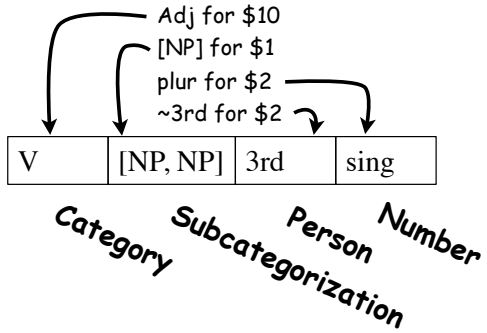
Priority invariant



$r \leq p.q$

Optimality

Constraints on what may be combined to form a phrase are defeasible. Optimality vectors:



Center-Embedding

This is the rat that ate the cheese.

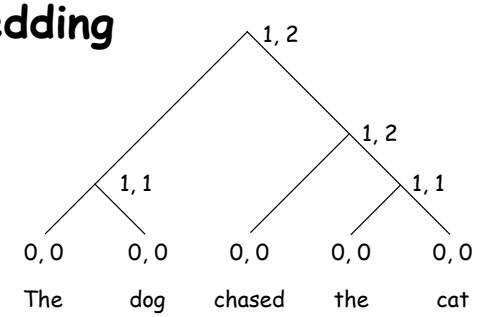
This is the cheese that the rat ate.

This is the cat that bit the rat that ate the cheese.

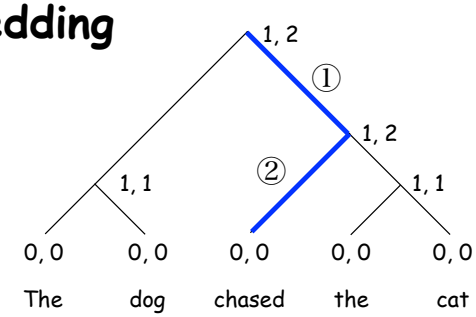
This is the cheese that the rat that the cat bit ate.

This is the cheese that [the rat that [the cat that [the dog chased] bit] ate].

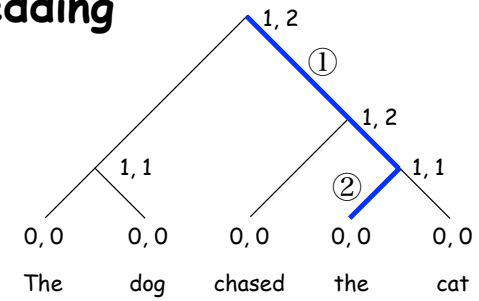
Center Embedding



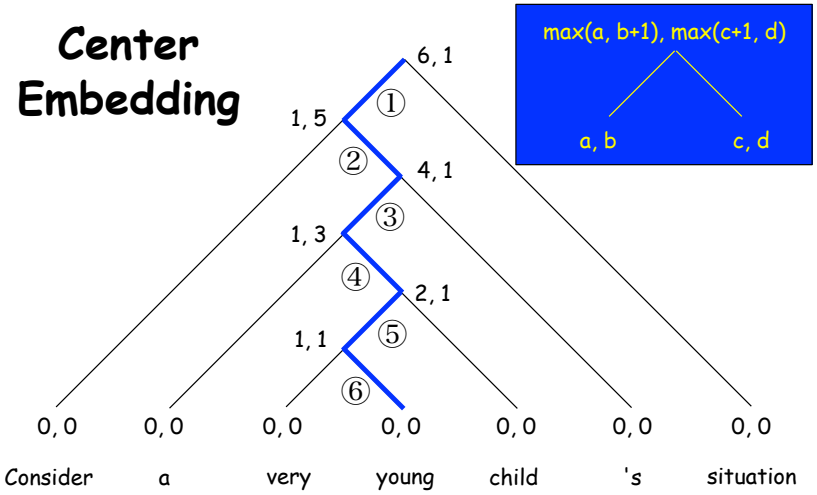
Center Embedding



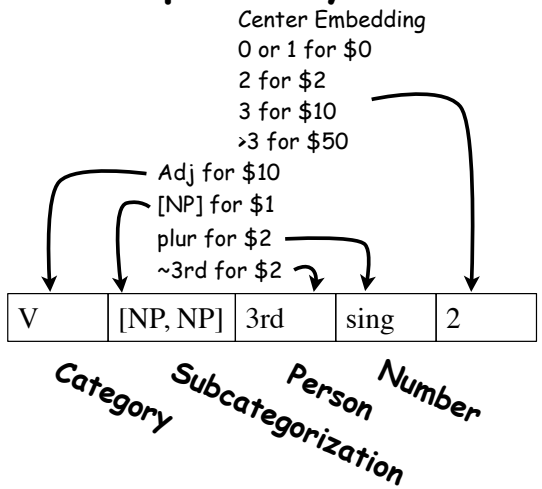
Center Embedding



Center Embedding



Optimality



Best First

approximates breadth first because longer phrases are much less probable and probabilities of phrases of different lengths are incomparable. Likewise for most other well-behaved figures of merit.

Merging Right-Hand Sides

Grammar might have rules

X → A G H P

X → B G H P

Could end up with both of these in chart:

(2, X → A . G H P)

(2, X → B . G H P)

But these are now interchangeable: if one produces X then so will the other

To avoid this redundancy, can always use dotted rules of the form: X → ... G H P

Merging Right-Hand Sides

Similarly, grammar might have rules

X → A G H P

X → A G H Q

Could end up with both of these in chart:

(2, X → A . G H P) in column 5

(2, X → A . G H Q) in column 5

Not interchangeable, but we'll be processing them in parallel for a while …

Solution: write grammar as X → A G H (P|Q)

Merging Right-Hand Sides

Combining the two previous cases:

X → A G H P

X → A G H Q

X → B G H P

X → B G H Q

becomes

X → (A | B) G H (P | Q)

And often nice to write stuff like

NP → (Det | ε) Adj* N

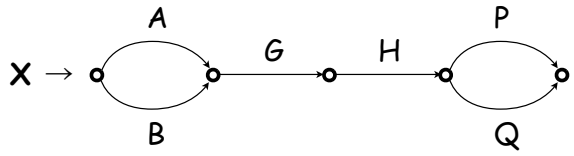
Merging Right-Hand Sides

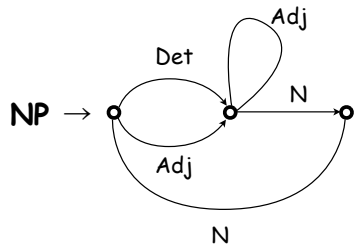
X → (A | B) G H (P | Q)

NP → (Det | ε) Adj* N

These are regular expressions!

Build their minimal DFAs:

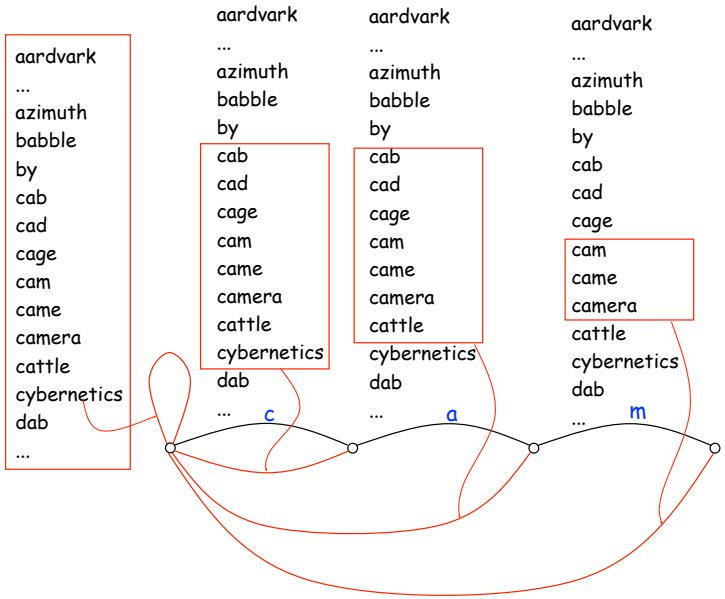




Indeed, all NP → rules can be combined into a single DFA!

NP → ADJP ADJP JJ JJ NN NNS
NP → ADJP DT NN
NP → ADJP JJ NN
NP → ADJP JJ NN NNS
NP → ADJP JJ NNS
NP → ADJP NN
NP → ADJP NN NN
NP → ADJP NN NNS
NP → ADJP NNS
NP → ADJP NPR
NP → ADJP NPRS
NP → DT
NP → DT ADJP
NP → DT ADJP , JJ NN
NP → DT ADJP ADJP NN
NP → DT ADJP JJ JJ NN
NP → DT ADJP JJ NN
NP → DT ADJP JJ NN NN

etc.

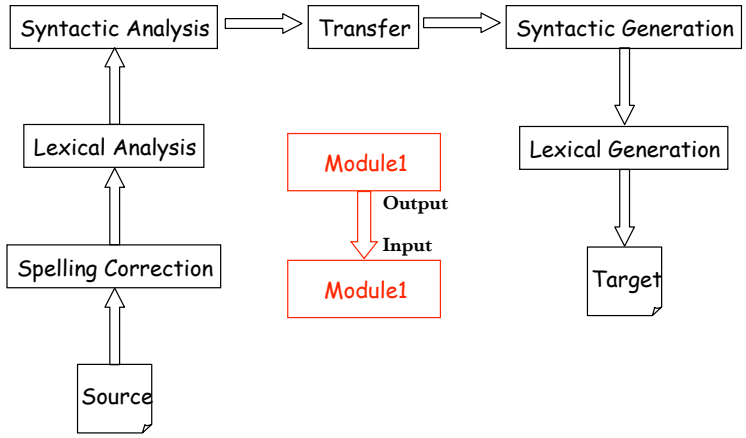


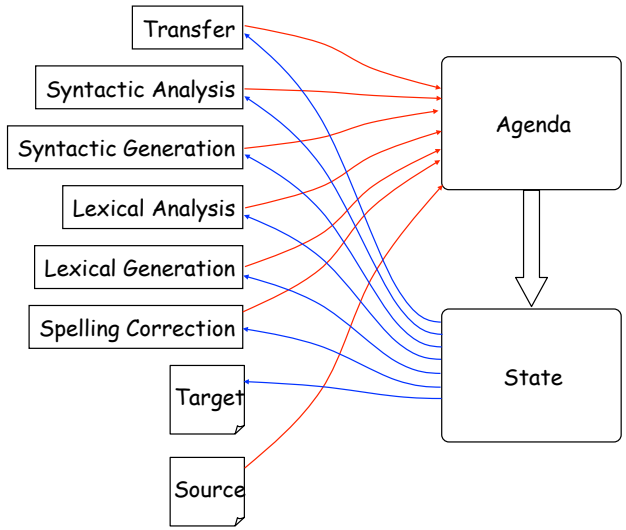

```
class Chart

  def initialize(agenda, monitor, filters=[])
  def reset
  def get_vertex(name)
  def schedule(task)
  def work_agenda
  def apply(active, inactive, new_pos)
  def each_vertex
  def sorted_vertices
  def show

end
```

Modular Design—Translation



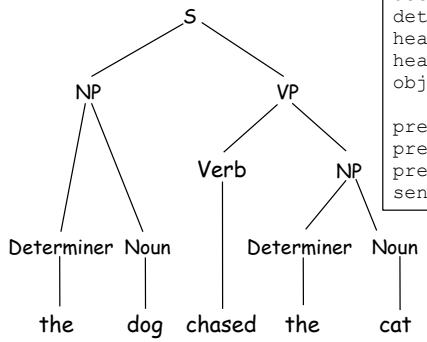



```
class Node
  attr_reader :name
  def initialize(name, daughters=[])
    @name = name
    @daughters = daughters
  end
end
```

```
det =  Node.new('the')
det =  Node.new('Determiner', [det])
head = Node.new('dog')
head = Node.new('Noun', [head])
subj = Node.new('NP', [det, head])

det =  Node.new('the')
det =  Node.new('Determiner', [det])
head = Node.new('cat')
head = Node.new('Noun', [head])
obj = Node.new('NP', [det, head])

pred = Node.new('chased')
pred = Node.new('Verb', [pred])
pred = Node.new('VP', [pred, obj])
sent = Node.new('S', [subj, pred])
```



```
S
  NP
    Determiner
      the
    Noun
      dog
  VP
    Verb
      chased
    NP
      Determiner
        the
      Noun
        cat
```

```
def scan(level=0)

  yield level, self
  for d in @daughters
    d.scan(level+1) {|l, n| yield l, n}
  end

end
```

```
sent.scan {|level, node| puts "#{'  '*level}#{node.name}"}
```



```
class Task

  def initialize(node, level=0)
    @node = node
    @level = level
  end

  def exec
    puts "#{'  '*@level}#{@node.name}"
    @node.each_daughter{|d| yield Task.new(d, @level+1)}
  end

end
```

```
agenda = [Task.new(sent)]
while task = agenda.shift
  task.exec {|t| agenda.unshift(t)}
end
```

```
S
  VP
    NP
      Noun
        dog
      Determiner
        the
    Verb
      chased
  NP
    Noun
      dog
    Determiner
      the
```



```
class Task

  def initialize(node, level=[])
    @node = node
    @level = level
  end

  def exec

    puts "#{'  '*@level.length}#{@node.name}"
    i=0
    @node.each_daughter do |d|
      yield Task.new(d, @level+[i])
      i+=1
    end

  end

end
```

```
S
  NP
    Determiner
      the
    Noun
      dog
  VP
    Verb
      chased
    NP
      Determiner
        the
      Noun
        cat
```

```
agenda = [Task.new(sent)]
while task = agenda.shift
  task.exec do |new_task|
    (0...agenda.length).each do |i|
      agenda[i], new_task = new_task, agenda[i] \
        if (new_task.level <=> agenda[i].level) == -1
    end
    agenda.push << new_task
  end
end
```


A* (A-star)

- *** begins at a selected node. Applied to this node are the "cost" of entering this node (usually zero for the initial node). A* then estimates the distance to the goal node from the current node. This estimate and the cost added together are the heuristic which is assigned to the path leading to this node. The node is then added to a priority queue, often called "open".**
- **The algorithm then removes the next node from the priority queue (because of the way a priority queue works, the node removed will have the lowest heuristic). If the queue is empty, there is no path from the initial node to the goal node and the algorithm stops. If the node is the goal node, A* reconstructs and outputs the successful path and stops. This path reconstruction from the stored closed nodes (see below) means it is not necessary to store the path-so-far within each node.**

A* (A-star)

- If the node is not the goal node, new nodes are created for all admissible adjoining nodes; the exact way of doing this depends on the problem at hand. For each successive node, A* calculates the "cost" of entering the node and saves it with the node. This cost is calculated from the cumulative sum of costs stored with its ancestors, plus the cost of the operation which reached this new node.The algorithm also maintains a "closed" list of nodes which have been checked. If a newly generated node is already in this list with an equal or lower cost, no further processing is done on that node or with the path associated with it. If a node in the closed list matches the new one, but has been stored with a higher cost, it is removed from the closed list, and processing continues on the new node.

A* (A-star)

- Next, an estimate of the new node's distance to the goal is added to the cost to form the heuristic for that node. This is then added to the "open" priority queue, unless an identical node with lesser or equal heuristic is found there.Once the above three steps have been repeated for each new adjoining node, the original node taken from the priority queue is added to the "closed" list. The next node is then popped from the priority queue and the process is repeated.

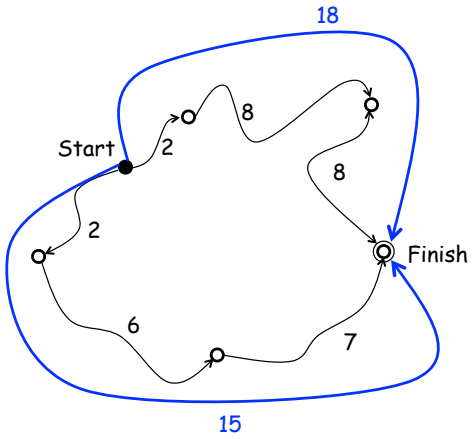
A*—Admissibility

- There is an intuitive explanation of why A* is both admissible and considers fewer nodes than any other admissible search algorithm. The intuition rests on the recognition that A* has an "optimistic" estimate of the cost of a path through every node that it considers -- optimistic in that the true cost of a path through that node to the goal will be at least as great as the estimate. But, critically, as far as A* "knows", that optimistic estimate might be achievable.
- When A* terminates its search, it by definition has found a path whose actual cost is lower than the estimated cost of any path through any open node. But since those estimates are optimistic, A* can safely ignore those nodes. In other words, A* will never overlook the possibility of a lower-cost path and so is admissible.
- Suppose now that some other search algorithm A terminates its search with a path whose actual cost is not less than the estimated cost of a path through some open node. Algorithm A cannot rule out the possibility, based on the heuristic information it has, that a path through that node might have a lower cost. So while A might consider fewer nodes than A*, it cannot be admissible. Accordingly, A* considers the fewest nodes of any admissible search algorithm that uses a no more accurate heuristic estimate.

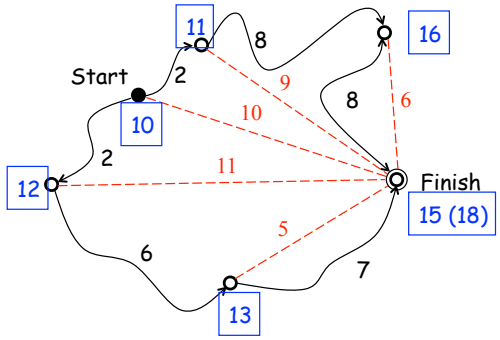
A*—Monotonicity

- If it can be guaranteed that the first path generated to any node is the shortest one, then the "closed list" is no longer necessary. Only one list of visited nodes needs to be maintained, so that nodes are not unnecessarily revisited. This guarantee can be made if the heuristic is not only admissible but monotonic, meaning that the difference between the heuristics of any two connected nodes does not overestimate the actual distance between those nodes. This is a form of the triangle inequality. Not all admissible heuristics are monotonic, but straight-line distance on a map is.

A* Search



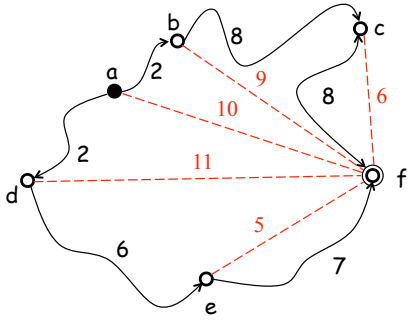
A* Search



A* Search

Open

➔ a 0 10 10

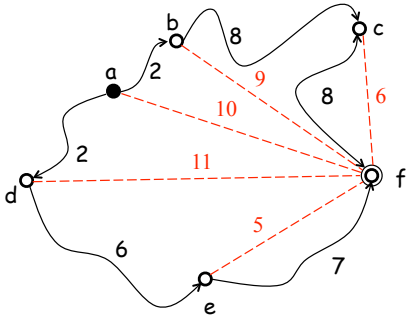


Closed

A* Search

Open

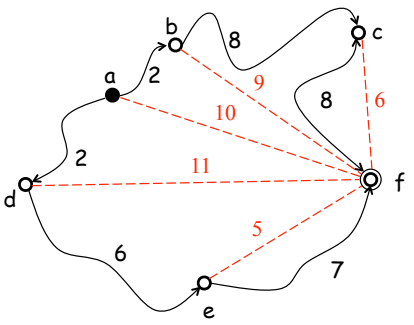
➔ b 2 9 11
d 2 11 13



Closed

a 0 10 10

A* Search



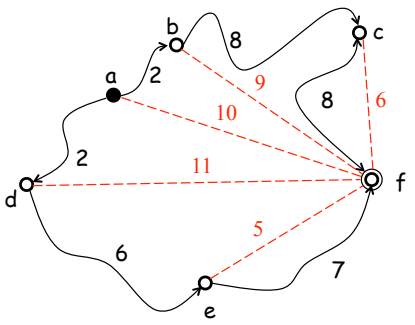
Open

➔	d	2	11	13
	c	10	6	16

Closed

a	0	10	10
b	2	9	11

A* Search



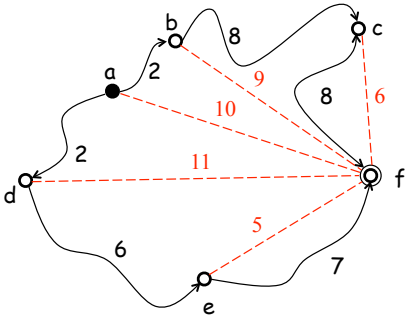
Open

➔	e	8	5	13
	c	10	6	16

Closed

a	0	10	10
b	2	9	11
d	2	11	13

A* Search



Open

➔	f	15	0	15
	c	10	6	16

Closed

a	0	10	10
b	2	9	11
d	2	11	13
e	8	5	13

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	1 V . drank
	1 V . snorted

predict

- Every .VP adds all VP → … rules again.
- Before adding a rule, check it's not a duplicate.
- **Slow** if there are > 700 VP → … rules, so what will you do in Homework 3?

1-word lookahead would help

0 Papa 1	ate
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	~~1 V . drank~~
	~~1 V . snorted~~
	~~1 P . with~~

No point in adding words other than ate

0 Papa 1	
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	0 NP NP . PP
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	1 PP . P NP
0 Det . a	1 V . ate
	1 V . drank
	1 V . snorted

predict

- .V makes us add all the verbs in the vocabulary!
- **Slow** – we'd like a shortcut.

With Left-Corner Filter

0 Papa 1	ate
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	
0 NP . Papa	
0 Det . the	
0 Det . a	

attach

PP can't start with ate

Birth control – now we won't predict
1 PP . P NP
1 PP . ate
either!

Need to know that ate can't start PP

Take closure of all categories that it does start

1-word lookahead would help

0 Papa 1	ate
0 ROOT . S	0 NP Papa .
0 S . NP VP	0 S NP . VP
0 NP . Det N	~~0 NP NP . PP~~
0 NP . NP PP	1 VP . V NP
0 NP . Papa	1 VP . VP PP
0 Det . the	~~1 PP . P NP~~
0 Det . a	1 V . ate
	~~1 V . drank~~
	~~1 V . snorted~~
	~~1 P . with~~

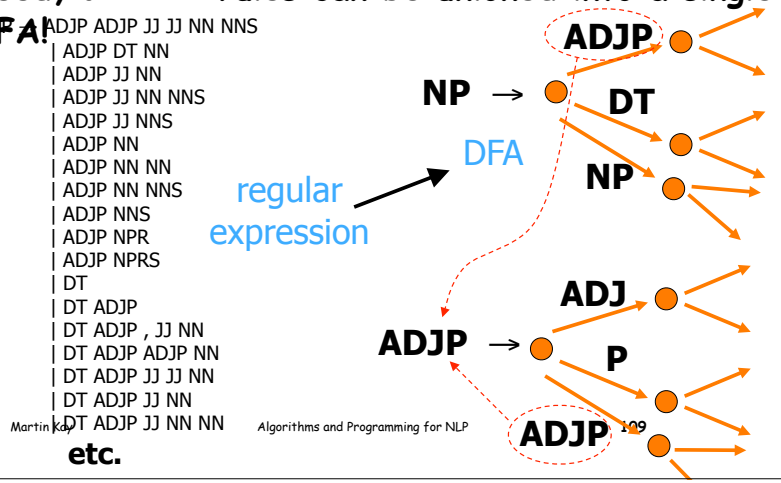
In fact, no point in adding any constituent that can't start with ate
Don't bother adding PP, P, etc.

No point in adding words other than ate

Merging Right-Hand Sides

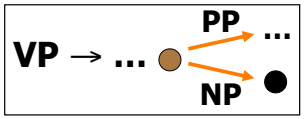
Indeed, all **NP →** rules can be unioned into a single DFA!

ADJP ADJP JJ JJ NN NNS
| ADJP DT NN
| ADJP JJ NN
| ADJP JJ NN NNS
| ADJP JJ NNS
| ADJP NN
| ADJP NN NN
| ADJP NN NNS
| ADJP NNS
| ADJP NPR
| ADJP NPRS
| DT
| DT ADJP
| DT ADJP , JJ NN
| DT ADJP ADJP NN
| DT ADJP JJ JJ NN
| DT ADJP JJ NN
| DT ADJP JJ NN NN
etc.



Earley's Algorithm on DFAs

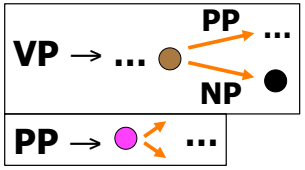
- What does Earley's algorithm now look like?

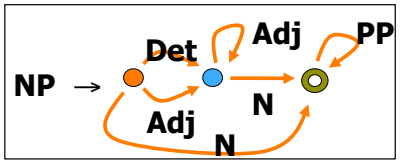


Column 4
...
(2, ●) predict

Earley's Algorithm on DFAs

- What does Earley's algorithm now look like?

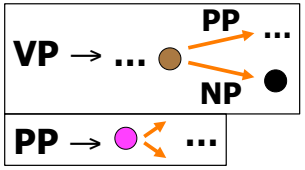


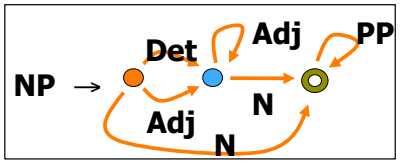


Column 4
...
(2, ●) predict
(4, ●)
(4, ●)

Earley's Algorithm on DFAs

- What does Earley's algorithm now look like?

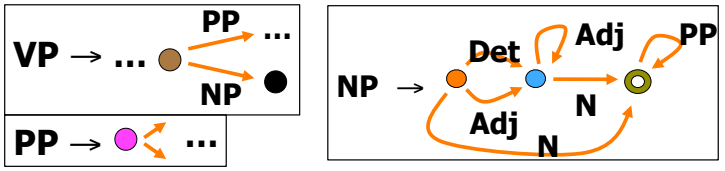




Column 4	Column 5	...	Column 7
...	...		
(2, ●)			(4, ○) predict or attach?
(4, ●)			
(4, ●) →	(4, ●)		

Earley's Algorithm on DFAs

- What does Earley's algorithm now look like?



Column 4	Column 5	...	Column 7
...	...		
(2, ●)			(4, ○)
(4, ●)			(7, ●)
(4, ●)	(4, ●)		(2, ●)

predict or attach?

Both!

Preprocessing

- First "tag" the input with parts of speech:
  - Guess the correct preterminal for each word, using faster methods we'll learn later
  - Now only allow one part of speech per word
  - This eliminates a lot of crazy constituents!
  - But if you tagged wrong you could be hosed

Raise the stakes:

Merging Right-Hand Sides

Indeed, all **NP → rules** can be unioned into a single DFA!

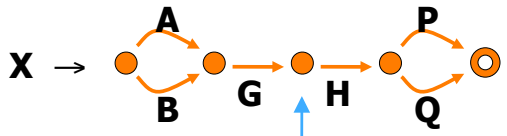
NP → ADJP ADJP JJ JJ NN NNS
NP → ADJP DT NN
NP → ADJP JJ NN
NP → ADJP JJ NN NNS
NP → ADJP JJ NNS
NP → ADJP NN
NP → ADJP NN NN
NP → ADJP NN NNS
NP → ADJP NNS
NP → ADJP NPR
NP → ADJP NPRS
NP → DT
NP → DT ADJP
NP → DT ADJP , JJ NN
NP → DT ADJP ADJP NN
NP → DT ADJP JJ JJ NN
NP → DT ADJP JJ NN
NP → DT ADJP JJ NN NN

Merging Right-Hand Sides

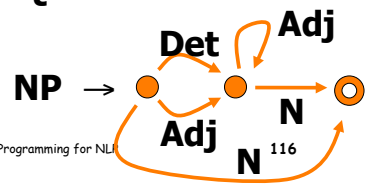
X → (A | B) G H (P | Q)

NP → (Det | ε) Adj* N

- These are regular expressions!
- Build their minimal DFAs:



- Automaton states replace dotted rules (**X → A G . H P**)



Merging Right-Hand Sides

- Combining the two previous cases:
  X → A G H P
  X → A G H Q
  X → B G H P
  X → B G H Q

becomes

  X → (A | B) G H (P | Q)

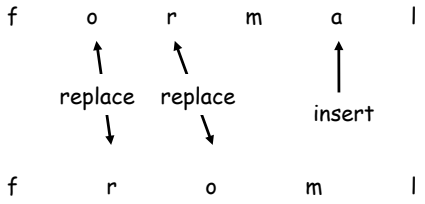
- And often nice to write stuff like
  NP → (Det | ε) Adj* N

Merging Right-Hand Sides

- Similarly, grammar might have rules
  X → A G H P
  X → A G H Q
- Could end up with both of these in chart:
  (2, X → A . G H P) in column 5
  (2, X → A . G H Q) in column 5
- Not interchangeable, but we'll be processing them in parallel for a while …
- Solution: write grammar as **X → A G H (P|Q)**

Merging Right-Hand Sides

- Grammar might have rules
  X → A G H P
  X → B G H P
- Could end up with both of these in chart:
  (2, X → A . G H P) in column 5
  (2, X → B . G H P) in column 5
- But these are now interchangeable: if one produces X then so will the other
- To avoid this redundancy, can always use dotted rules of this form: **X → … G H P**



A* Edit Distance

to goal

from start

	f	r	o	m	l
f	1 0	2 1			
o	0 1	1 0	2 1		
r	1 2	0 2	1 1	2 2	
m		1 3	0 2	1 2	
a				0 3	1 2
l					0 3

Make initial chart and agenda.

Repeat until agenda is empty:

Take first arc from agenda.

Add arc to chart. (Only do this if edge is not already on the chart!)

Use the fundametal rule to combine this arc with arcs from the chart. Any edges obtained in this way should be added to the agenda.

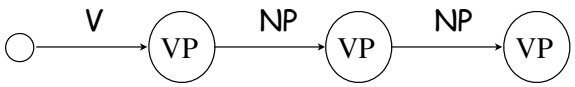
Make hypotheses (i.e., active edges) about new constituents

Representation of Rules

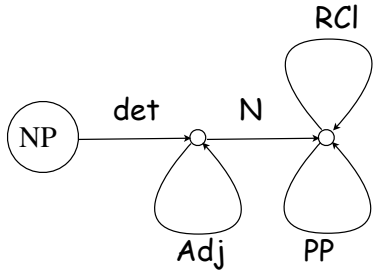
VP → V NP NP

VP → V NP

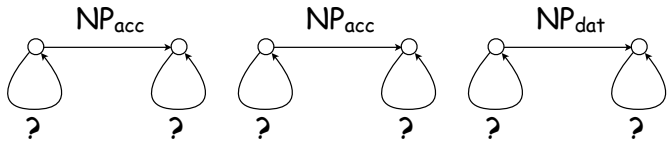
VP → V



Representation of Rules



Representation of Rules



NP– V_{fin} NP– NP– NP– V_{nonfin}