



# Recitation 9

## **CTC Decoding and Beam Search**

Soumya Empran & Ameya Mahabaleshwarkar

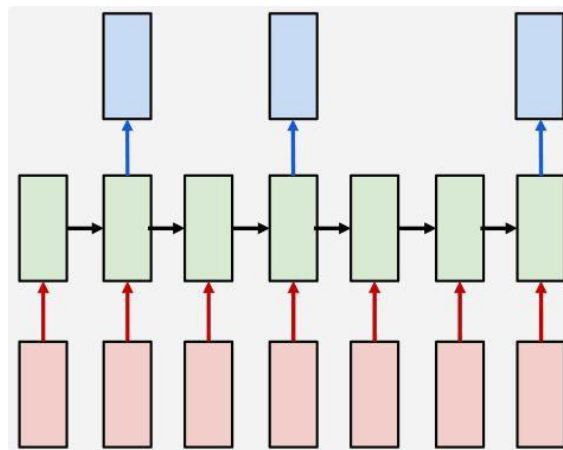
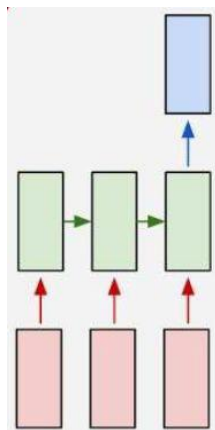


# Sequence to Sequence Modeling

- Problem:
  - Input Sequence:  $X_1 \dots X_n$
  - Output Sequence:  $Y_1 \dots Y_m$
- $|X| \neq |Y|$

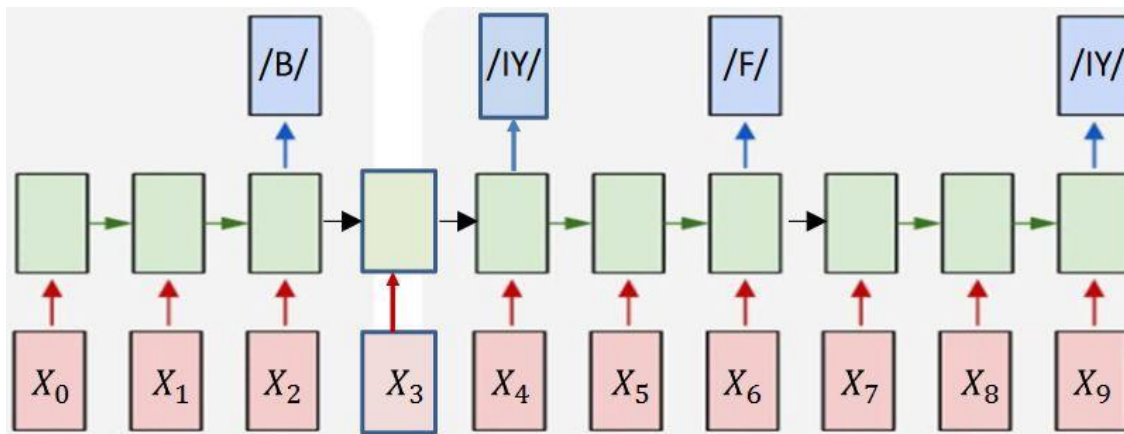
## HW3P2 Problem: Sequence to Sequence with Order Synchrony

- In HW1P2, we utilized sequence classification for phoneme recognition. We can manage this problem by applying a variant using recurrent nets.
- Left: Sequence of inputs produces a single output; Right: How???

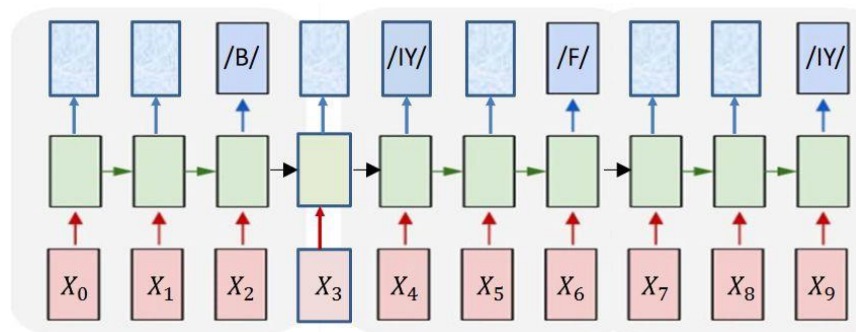


## New: Complex Problem - Training

- Objective: Given a sequence of inputs, asynchronously output a sequence of symbols
  - Concatenation of many copies of the simple model in the previous slide



- In the previous model, we ignored intermediate steps. However, we can exploit the untagged inputs and assume the same output.

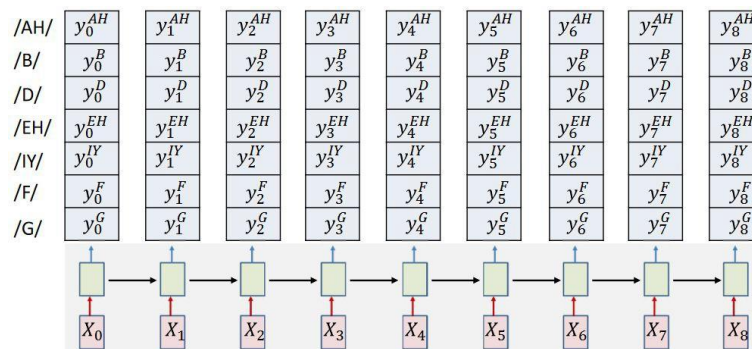
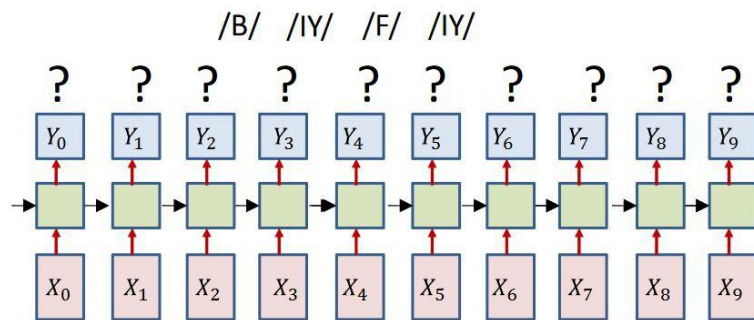


- How do we know when to output symbols?
  - Apply our ideas from HW1P2:
    - At each time in the network outputs a probability for each output symbol given all inputs until that time.
    - The most likely symbol sequence given the inputs. **How?**

## Lecture will discuss computing Divergence

- Possible Solutions

- **Solution 1:** Simply select the most probable symbol at each time. Merge adjacent repeated symbols, and place the actual emission of the symbol in the final instant.
  - **Issue 1:** This isn't the most probable sequence of symbols
  - **Issue 2:** Cannot distinguish between an extended symbol and repetitions of the symbol
- **Solution 2:** Impose external constraints on what sequences are allowed
  - **Issue 1:** A suboptimal decode that actually finds the most likely time-synchronous output sequence. Will be discussed in lecture.



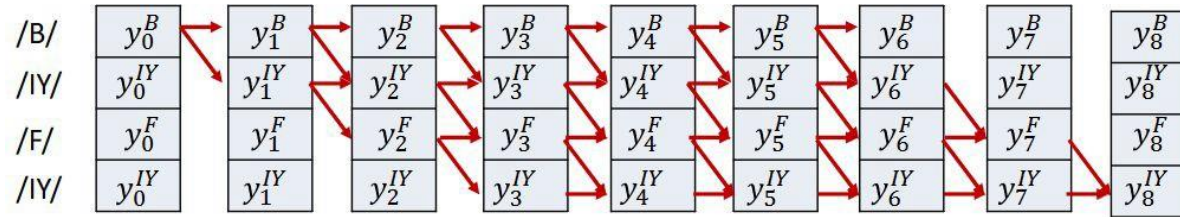
- Overall Solution:

- Apply both previous solutions

- At each time the network outputs a probability for each output symbol

- Block out all rows that do not include symbols from the target sequence

- Compose a graph such that every path in the graph from source to sink represents a valid alignment



- Find the most probable sequence of symbols using the graph above
  - Edge scores have a probability of 1
  - Nodes scores are probabilities resulting from the neural network

Lecture will discuss how to find the most probable sequence given the graph and how to compute the divergence once we get the most probable sequence

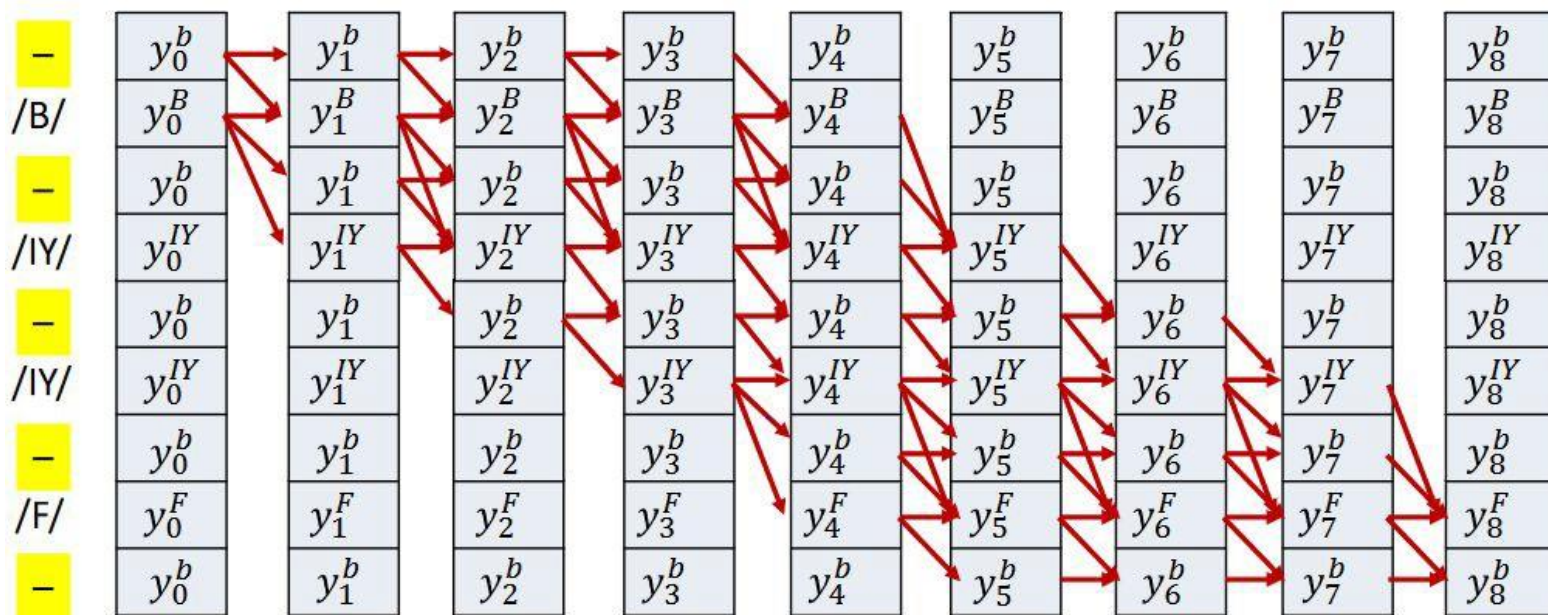




# Repetition Issue and Solution

- We have a decode:
  - R R R O O O O D
  - Is this the symbol sequence ROD or ROOD?
- Introduce an explicit extra symbol which serves to separate discrete versions of a symbol (Blank)
  - RRR---OO---DDD = ROD
  - -RR-R---OO---D-DD = RRODD
- The label recognized by the network must now include the extra blank symbol that will need to be trained

# Final Graph



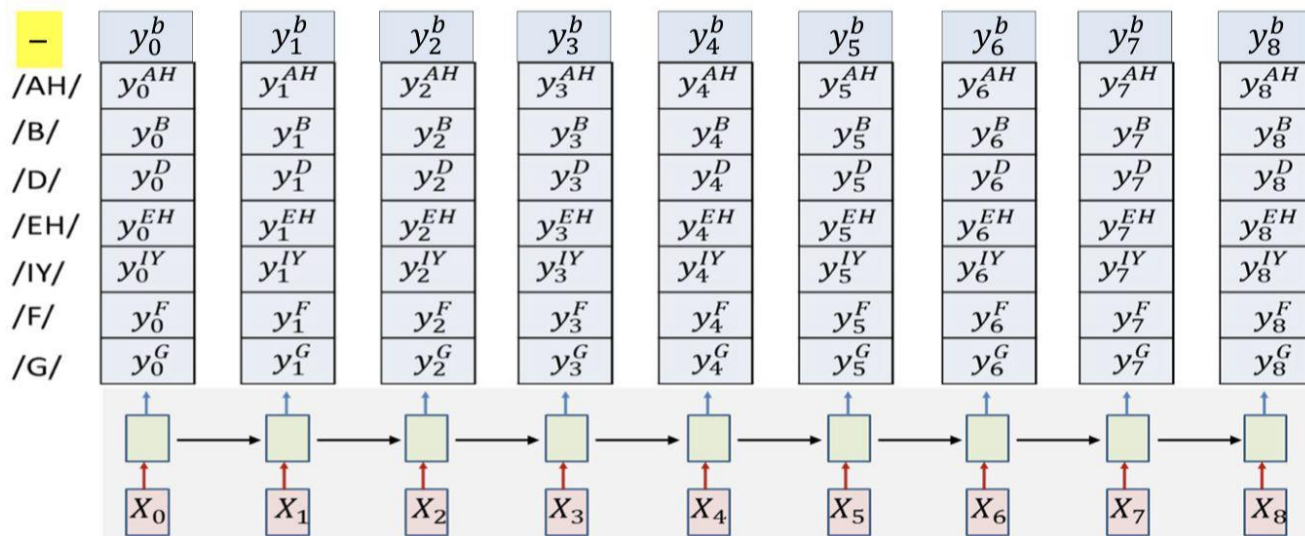


# CTC - Training Procedure

1. Setup Network
  - a. Many LSTM
2. Initialize network with a Blank Symbol
3. Pass training instances through network to obtain probabilities for all labels/symbols
4. Construct graph on previous page
5. Forward and Backward Algorithm - **Lecture**
6. Compute Divergence - **Lecture**
7. Update Parameters

**Connectionist Temporal Classification**

# The forward output



# Greedy Search

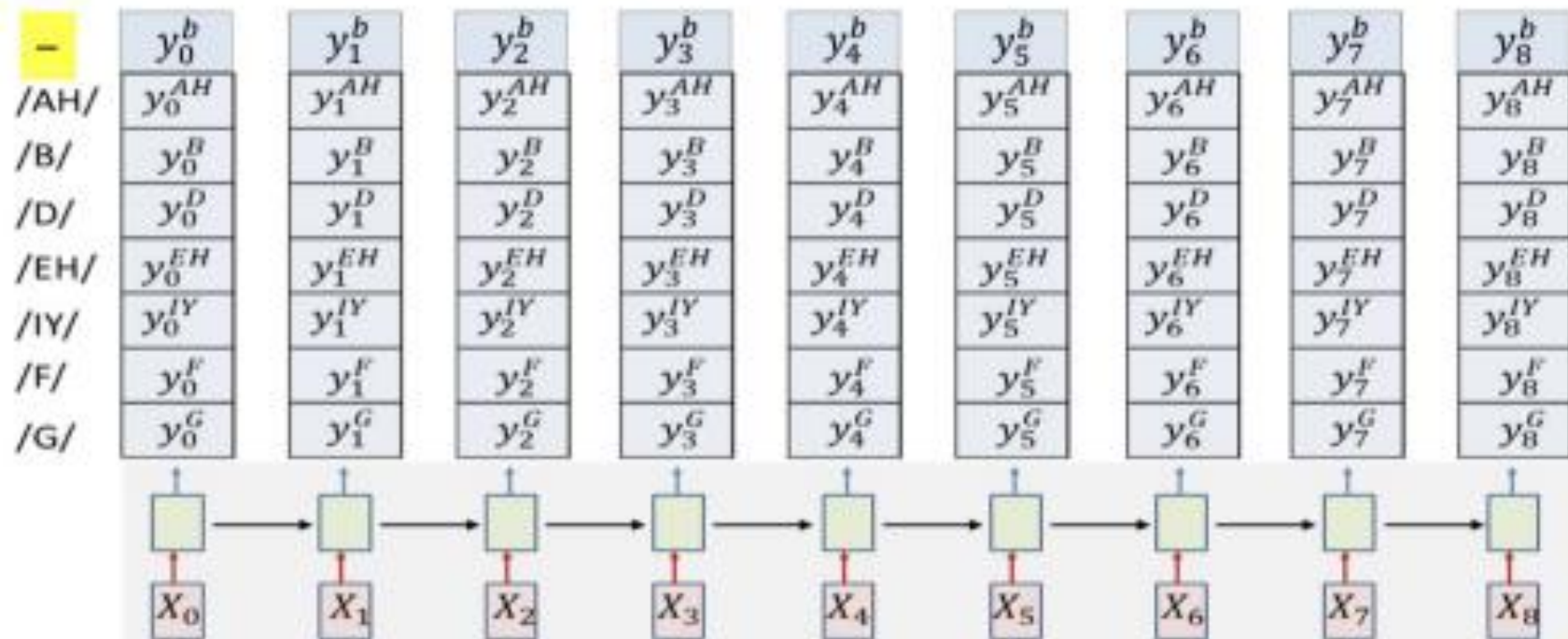
- Greedy Search is an easy-to-implement option for CTC decoding at inference time
- Greedy Search simply selects the most probable time step at each time-step
- Although this method is easy to implement and fast, it has the disadvantage of missing out on high-probability (score) overall paths due to its greedy search

# Exhaustive Search

- An alternative to the “short-sighted” Greedy Search, we can conduct a search over all possible paths, and then select the best possible output
- This method will guarantee decoding an optimal path / sequence
- The disadvantage is that Exhaustive Search will be exponential in output symbol sequences, and hence is not a feasible option

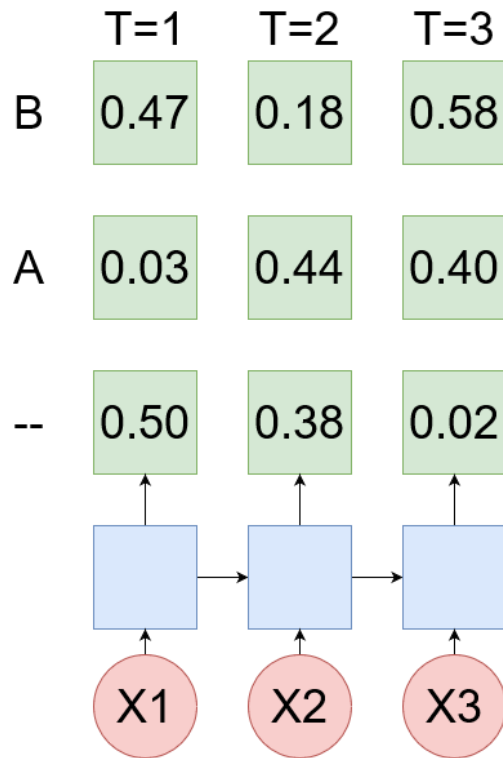
# Beam Search

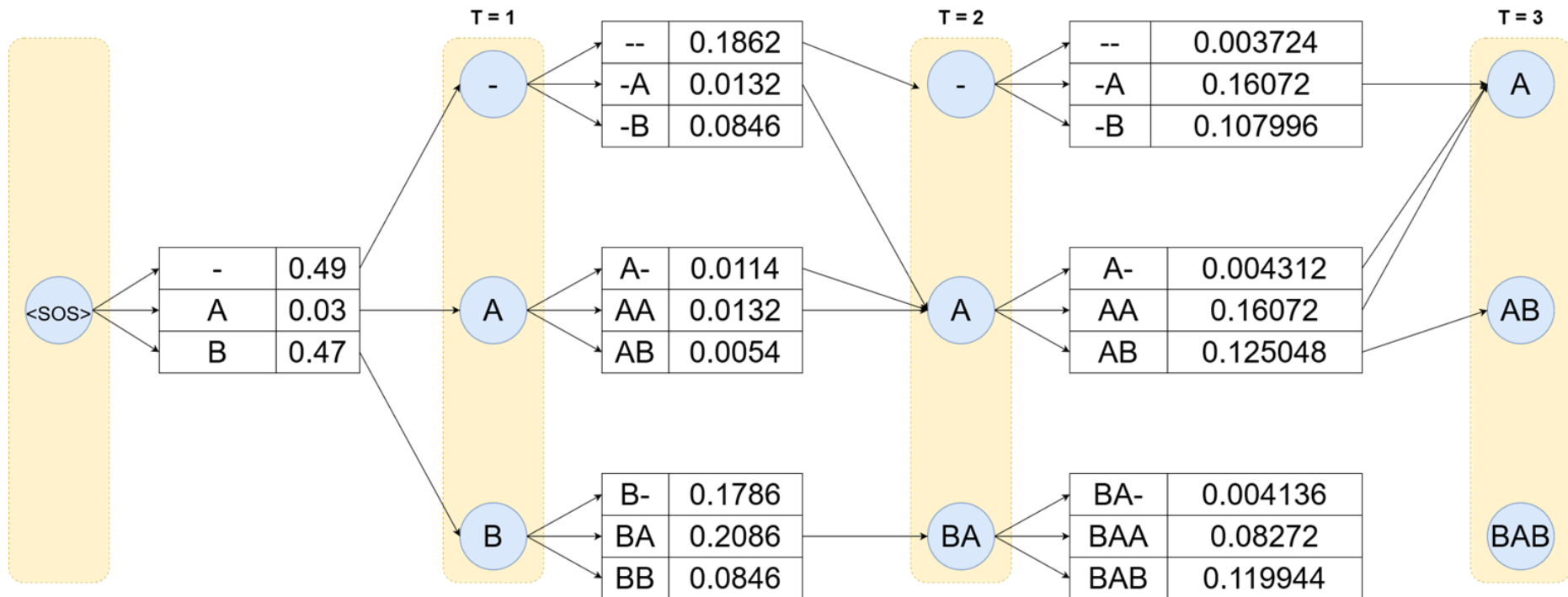
- To have better decoding than Greedy Search, but keep the method feasible at the same time, we can choose to “explore” top-k paths at each time-step
- By exploring more than one most-probable output sequences at each time-step, we will reach a sub-optimal path that is likely to be better than the Greedy Search strategy
- By limiting our exploration options to a specific Beam Width -  $k$ , we also ensure that the computation is tractable, as opposed to the Exhaustive Search strategy





Let's use some actual values





# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
            NewBlankPathScore, NewPathScore, BeamWidth)

    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
        PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
        PathsWithTerminalSymbol, SymbolSet, y[:,t])

end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore
    NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []
```

```
# First time instant: Initialize paths with each of the symbols,  
# including blank, using score at time t=1  
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =  
    InitializePaths(SymbolSet, y[:,0])
```

```
# Subsequent time steps
```

```
for t = 1:T
```

```
    # Prune the collection down to the BeamWidth
```

```
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =  
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,  
              NewBlankPathScore, NewPathScore, BeamWidth)
```

```
    # First extend paths by a blank
```

```
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,  
                                                                    PathsWithTerminalSymbol, y[:,t])
```

```
    # Next extend paths by a symbol
```

```
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,  
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])
```

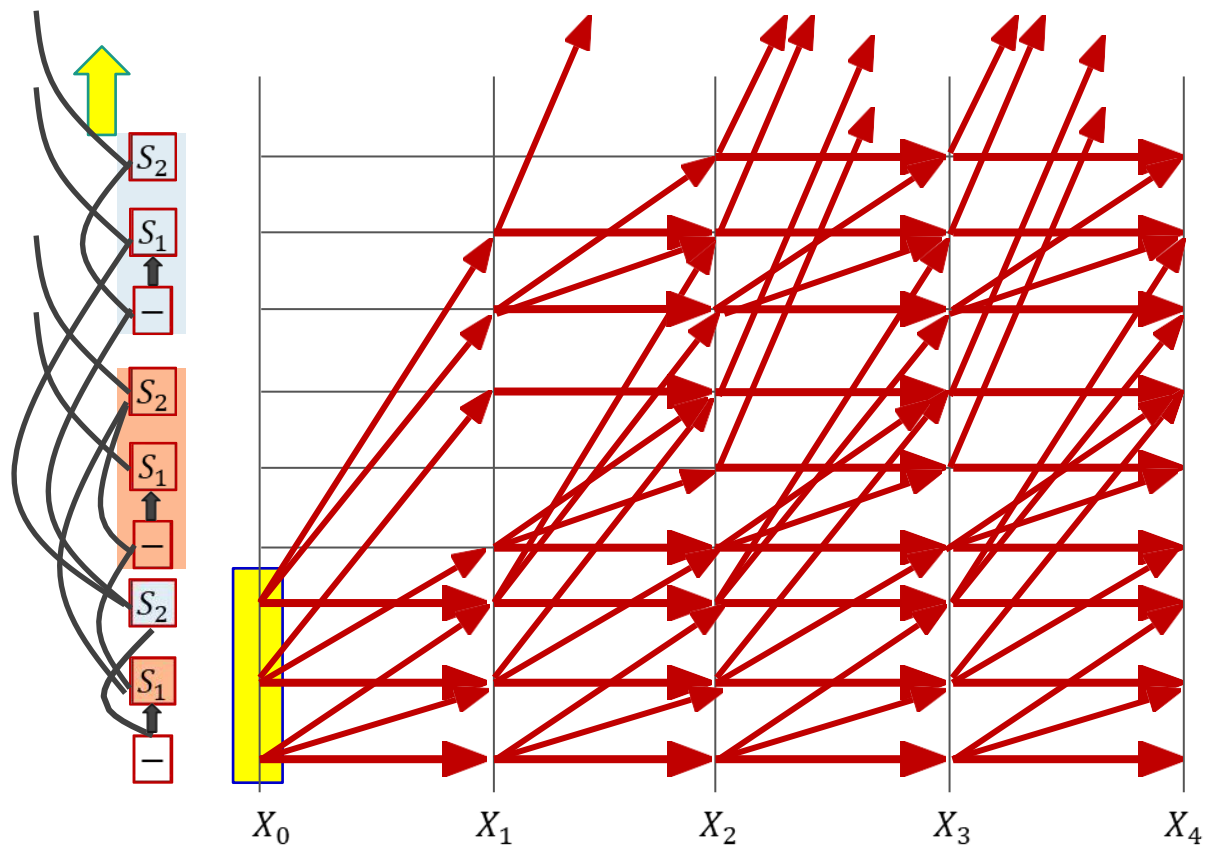
```
end
```

```
# Merge identical paths differing only by the final blank
```

```
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore  
                                                    NewPathsWithTerminalSymbol, NewPathScore)
```

```
# Pick best path
```

```
BestPath = argmax(FinalPathScore) # Find the path with the best score
```



## BEAM SEARCH InitializePaths: FIRST TIME INSTANT

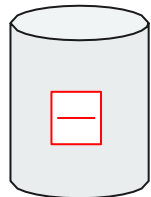
```
function InitializePaths(SymbolSet, y)

InitialBlankPathScore = [], InitialPathScore = []
# First push the blank into a path-ending-with-blank stack. No symbol has been invoked yet
path = null
InitialBlankPathScore[path] = y[blank] # Score of blank at t=1
InitialPathsWithFinalBlank = {path}

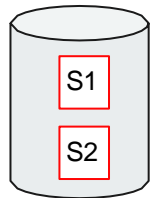
# Push rest of the symbols into a path-ending-with-symbol stack
InitialPathsWithFinalSymbol = {}
for c in SymbolSet # This is the entire symbol set, without the blank
    path = c
    InitialPathScore[path] = y[c] # Score of symbol c at t=1
    InitialPathsWithFinalSymbol += path # Set addition
end

return InitialPathsWithFinalBlank, InitialPathsWithFinalSymbol,
    InitialBlankPathScore, InitialPathScore
```

InitialPathWithFinalBlank



InitialPathWithFinalSymbols



# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)

    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])

end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore,
                                                    NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

We will visit this routine  
after discussing the rest of  
the loop  
(to avoid confusion)

# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)

    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])

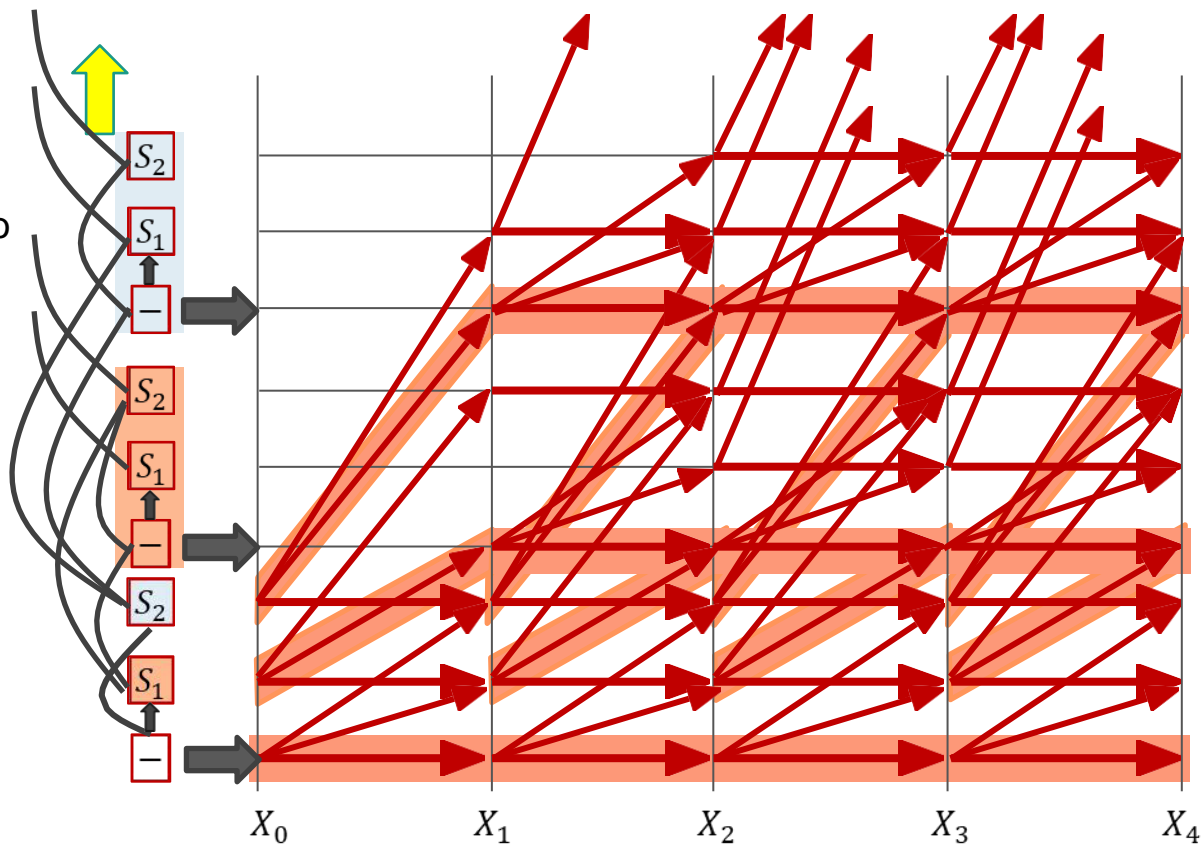
end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore
                                                    NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```



Only transitions into nodes on the rows corresponding to blanks



## BEAM SEARCH: Extending with blanks

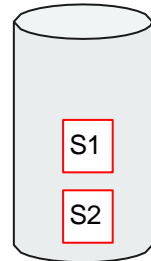
Global PathScore, BlankPathScore

```
function ExtendWithBlank(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalBlank = {}
    UpdatedBlankPathScore = []
    # First work on paths with terminal blanks
    #(This represents transitions along horizontal trellis edges for blanks)
    for path in PathsWithTerminalBlank:
        # Repeating a blank doesn't change the symbol sequence
        UpdatedPathsWithTerminalBlank += path    # Set addition
        UpdatedBlankPathScore[path] = BlankPathScore[path]*y[blank]
    end

    # Then extend paths with terminal symbols by blanks
    for path in PathsWithTerminalSymbol:
        # If there is already an equivalent string in UpdatesPathsWithTerminalBlank
        # simply add the score. If not create a new entry
        if path in UpdatedPathsWithTerminalBlank
            UpdatedBlankPathScore[path] += Pathscore[path]* y[blank]
        else
            UpdatedPathsWithTerminalBlank += path    # Set addition
            UpdatedBlankPathScore[path] = PathScore[path] * y[blank]
        end
    end

    return UpdatedPathsWithTerminalBlank,
           UpdatedBlankPathScore
```

( only at t=1)  
UpdatedPathsWithTerminalBlank



# BEAM SEARCH: Extending with blanks

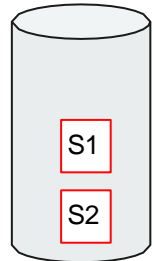
Global PathScore, BlankPathScore

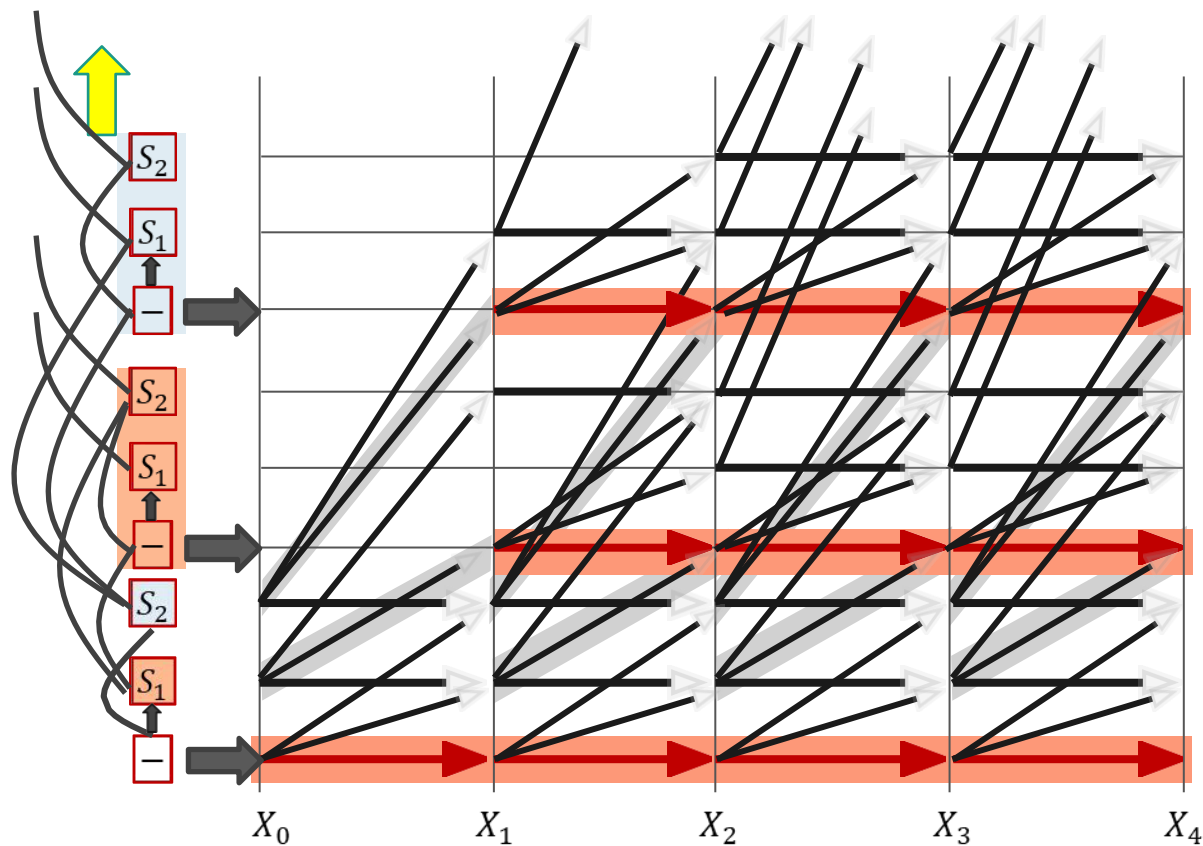
```
function ExtendWithBlank(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalBlank = {}
    UpdatedBlankPathScore = []
    # First work on paths with terminal blanks
    #(This represents transitions along horizontal trellis edges for blanks)
    for path in PathsWithTerminalBlank:
        # Repeating a blank doesn't change the symbol sequence
        UpdatedPathsWithTerminalBlank += path    # Set addition
        UpdatedBlankPathScore[path] = BlankPathScore[path]*y[blank]
    end

    # Then extend paths with terminal symbols by blanks
    for path in PathsWithTerminalSymbol:
        # If there is already an equivalent string in UpdatesPathsWithTerminalBlank
        # simply add the score. If not create a new entry
        if path in UpdatedPathsWithTerminalBlank
            UpdatedBlankPathScore[path] += Pathscore[path]* y[blank]
        else
            UpdatedPathsWithTerminalBlank += path    # Set addition
            UpdatedBlankPathScore[path] = PathScore[path] * y[blank]
        end
    end
end

return UpdatedPathsWithTerminalBlank,
        UpdatedBlankPathScore
```

( only at t=1)  
UpdatedPathsWithTerminalBlank





Transitions from  
"blank" lines  
to "blank" lines  
(which will all be  
horizontal edges)

## BEAM SEARCH: Extending with blanks

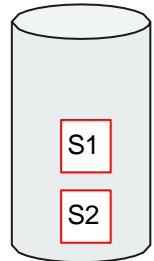
Global PathScore, BlankPathScore

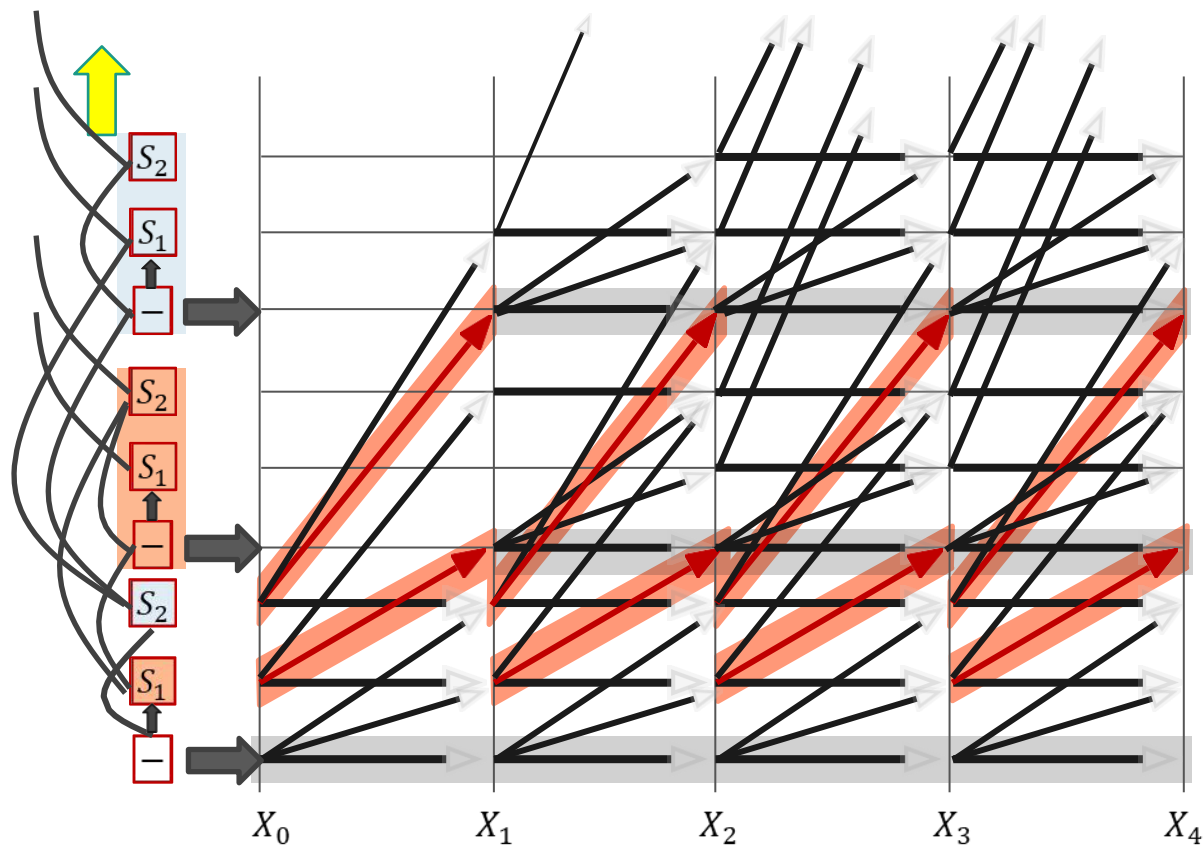
```
function ExtendWithBlank(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalBlank = {}
    UpdatedBlankPathScore = []
    # First work on paths with terminal blanks
    #(This represents transitions along horizontal trellis edges for blanks)
    for path in PathsWithTerminalBlank:
        # Repeating a blank doesn't change the symbol sequence
        UpdatedPathsWithTerminalBlank += path    # Set addition
        UpdatedBlankPathScore[path] = BlankPathScore[path]*y[blank]
    end

    # Then extend paths with terminal symbols by blanks
    for path in PathsWithTerminalSymbol:
        # If there is already an equivalent string in UpdatesPathsWithTerminalBlank
        # simply add the score. If not create a new entry
        if path in UpdatedPathsWithTerminalBlank
            UpdatedBlankPathScore[path] += Pathscore[path]* y[blank]
        else
            UpdatedPathsWithTerminalBlank += path    # Set addition
            UpdatedBlankPathScore[path] = PathScore[path] * y[blank]
        end
    end

    return UpdatedPathsWithTerminalBlank,
           UpdatedBlankPathScore
```

( only at t=1)  
UpdatedPathsWithTerminalBlank





Transitions from  
"symbol" lines  
to "blank" lines

# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)

    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

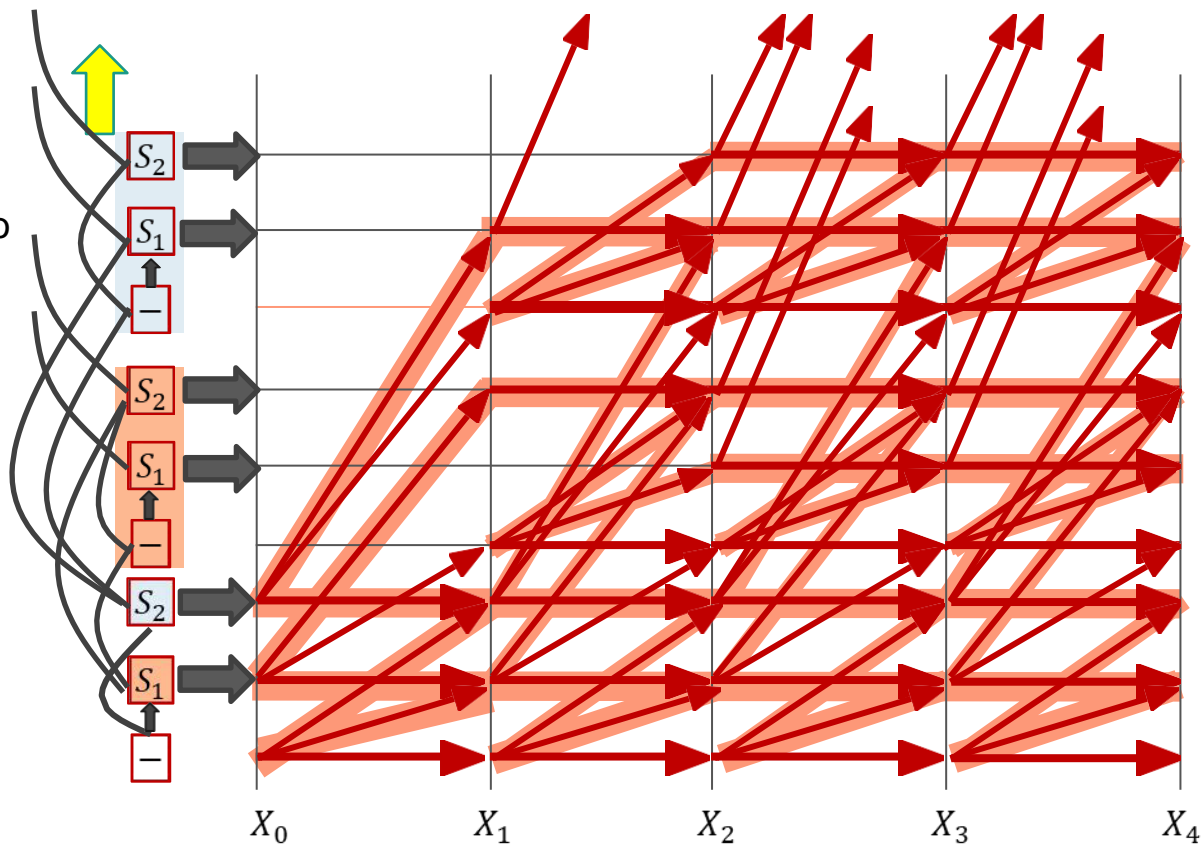
    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])

end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore
                                                    NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

Only transitions into nodes on the rows corresponding to non-blank symbols



(figure shows path extensions for only 2 time steps)



## BEAM SEARCH: Extending with symbols

Global PathScore, BlankPathScore

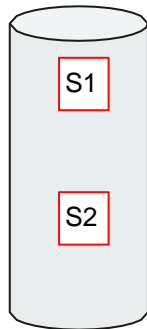
```
function ExtendWithSymbol(PathsWithTerminalBlank, PathsWithTerminalSymbol, SymbolSet, y)
    UpdatedPathsWithTerminalSymbol = {}
    UpdatedPathScore = []

    # First extend the paths terminating in blanks. This will always create a new sequence
    for path in PathsWithTerminalBlank:
        for c in SymbolSet: # SymbolSet does not include blanks
            newpath = path + c # Concatenation
            UpdatedPathsWithTerminalSymbol += newpath # Set addition
            UpdatedPathScore[newpath] = BlankPathScore[path] * y(c)
        end
    end

    # Next work on paths with terminal symbols
    for path in PathsWithTerminalSymbol:
        # Extend the path with every symbol other than blank
        for c in SymbolSet: # SymbolSet does not include blanks
            newpath = (c == path[end]) ? path : path + c # Horizontal transitions don't extend the sequence
            if newpath in UpdatedPathsWithTerminalSymbol: # Already in list, merge paths
                UpdatedPathScore[newpath] += PathScore[path] * y[c]
            else # Create new path
                UpdatedPathsWithTerminalSymbol += newpath # Set addition
                UpdatedPathScore[newpath] = PathScore[path] * y[c]
            end
        end
    end

    return UpdatedPathsWithTerminalSymbol, UpdatedPathScore
```

( only at t=1 )  
UpdatedPathsWithTerminalSymbol



# BEAM SEARCH: Extending with symbols

Global PathScore, BlankPathScore

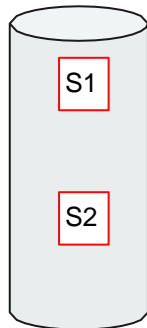
```
function ExtendWithSymbol(PathsWithTerminalBlank, PathsWithTerminalSymbol, SymbolSet, y)
    UpdatedPathsWithTerminalSymbol = {}
    UpdatedPathScore = []

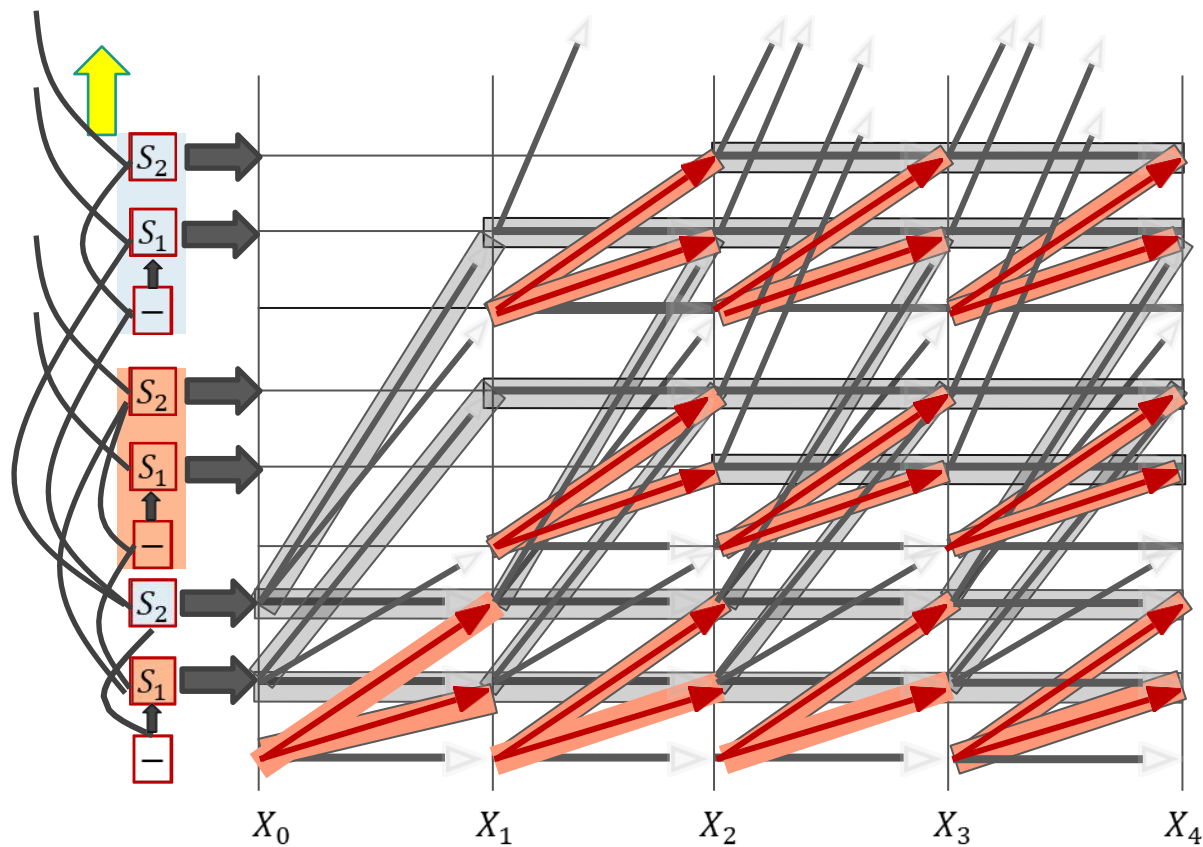
    # First extend the paths terminating in blanks. This will always create a new sequence
    for path in PathsWithTerminalBlank:
        for c in SymbolSet: # SymbolSet does not include blanks
            newpath = path + c # Concatenation
            UpdatedPathsWithTerminalSymbol += newpath # Set addition
            UpdatedPathScore[newpath] = BlankPathScore[path] * y(c)
        end
    end

    # Next work on paths with terminal symbols
    for path in PathsWithTerminalSymbol:
        # Extend the path with every symbol other than blank
        for c in SymbolSet: # SymbolSet does not include blanks
            newpath = (c == path[end]) ? path : path + c # Horizontal transitions don't extend the sequence
            if newpath in UpdatedPathsWithTerminalSymbol: # Already in list, merge paths
                UpdatedPathScore[newpath] += PathScore[path] * y[c]
            else # Create new path
                UpdatedPathsWithTerminalSymbol += newpath # Set addition
                UpdatedPathScore[newpath] = PathScore[path] * y[c]
            end
        end
    end

    return UpdatedPathsWithTerminalSymbol, UpdatedPathScore
```

( only at t=1 )  
UpdatedPathsWithTerminalSymbol





Transitions from  
"blank" lines  
to "symbol" lines

(figure shows path extensions for only 2 time steps)

## BEAM SEARCH: Extending with symbols

Global PathScore, BlankPathScore

```
function ExtendWithSymbol(PathsWithTerminalBlank, PathsWithTerminalSymbol, SymbolSet, y)
```

```
    UpdatedPathsWithTerminalSymbol = {}
```

```
    UpdatedPathScore = []
```

```
    # First extend the paths terminating in blanks. This will always create a new sequence
```

```
    for path in PathsWithTerminalBlank:
```

```
        for c in SymbolSet: # SymbolSet does not include blanks
```

```
            newpath = path + c # Concatenation
```

```
            UpdatedPathsWithTerminalSymbol += newpath # Set addition
```

```
            UpdatedPathScore[newpath] = BlankPathScore[path] * y(c)
```

```
        end
```

```
    end
```

```
    # Next work on paths with terminal symbols
```

```
    for path in PathsWithTerminalSymbol:
```

```
        # Extend the path with every symbol other than blank
```

```
        for c in SymbolSet: # SymbolSet does not include blanks
```

```
            newpath = (c == path[end]) ? path : path + c # Horizontal transitions don't extend the sequence
```

```
            if newpath in UpdatedPathsWithTerminalSymbol: # Already in list, merge paths
```

```
                UpdatedPathScore[newpath] += PathScore[path] * y[c]
```

```
            else # Create new path
```

```
                UpdatedPathsWithTerminalSymbol += newpath # Set addition
```

```
                UpdatedPathScore[newpath] = PathScore[path] * y[c]
```

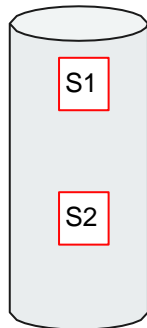
```
            end
```

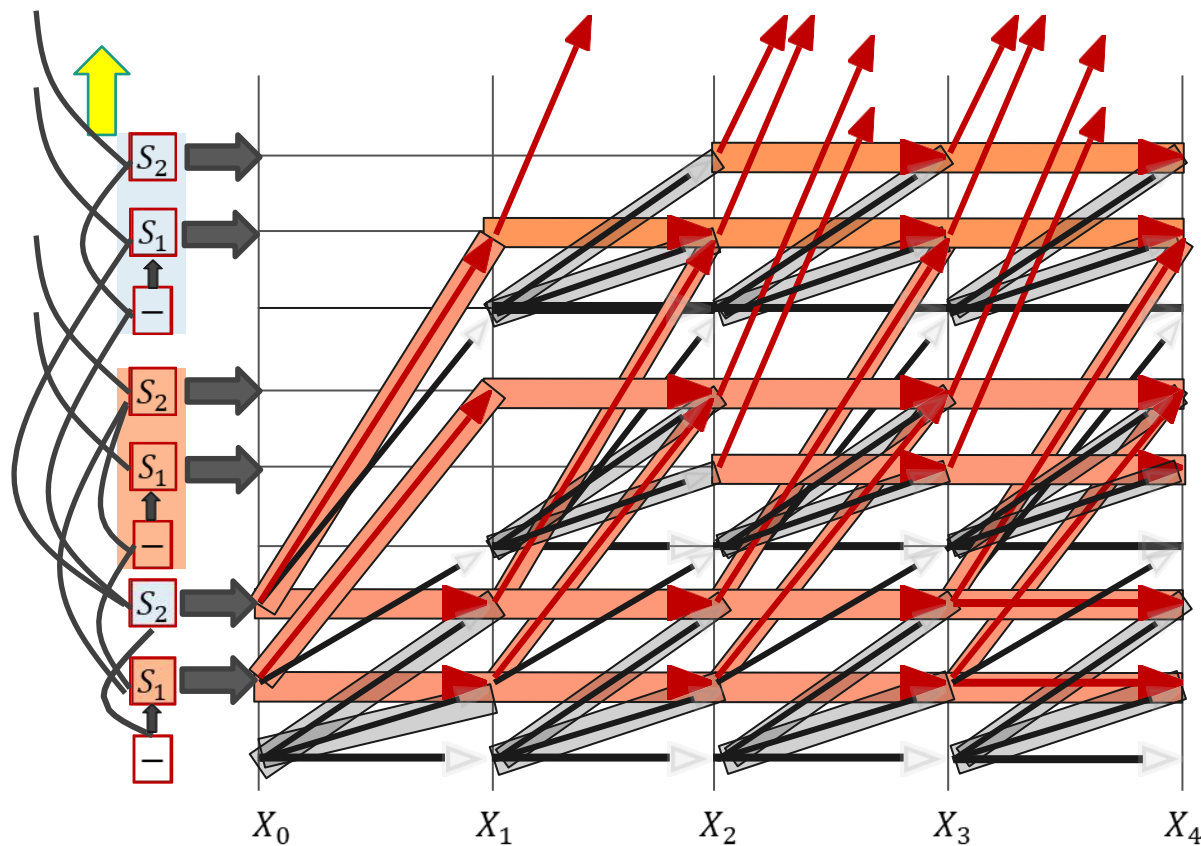
```
        end
```

```
    end
```

```
return UpdatedPathsWithTerminalSymbol, UpdatedPathScore
```

( only at t=1 )  
UpdatedPathsWlthTerminalSymbol





Transitions from  
"symbol" lines  
to "symbol" lines  
(including horizontal  
transitions)

(figure shows path extensions for only 2 time steps)

# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])
```

```
# Subsequent time steps
```

```
for t = 1:T
```

```
    # Prune the collection down to the BeamWidth
```

```
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)
```

```
    # First extend paths by a blank
```

```
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    BlankPathScore,
                                                                    PathsWithTerminalSymbol, PathScore)
```

```
    # Next extend paths by a symbol
```

```
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                NewBlankPathScore,
                                                                PathsWithTerminalSymbol, SymbolSet, y[t,:])
```

```
end
```

```
# Merge identical paths differing only by the final blank
```

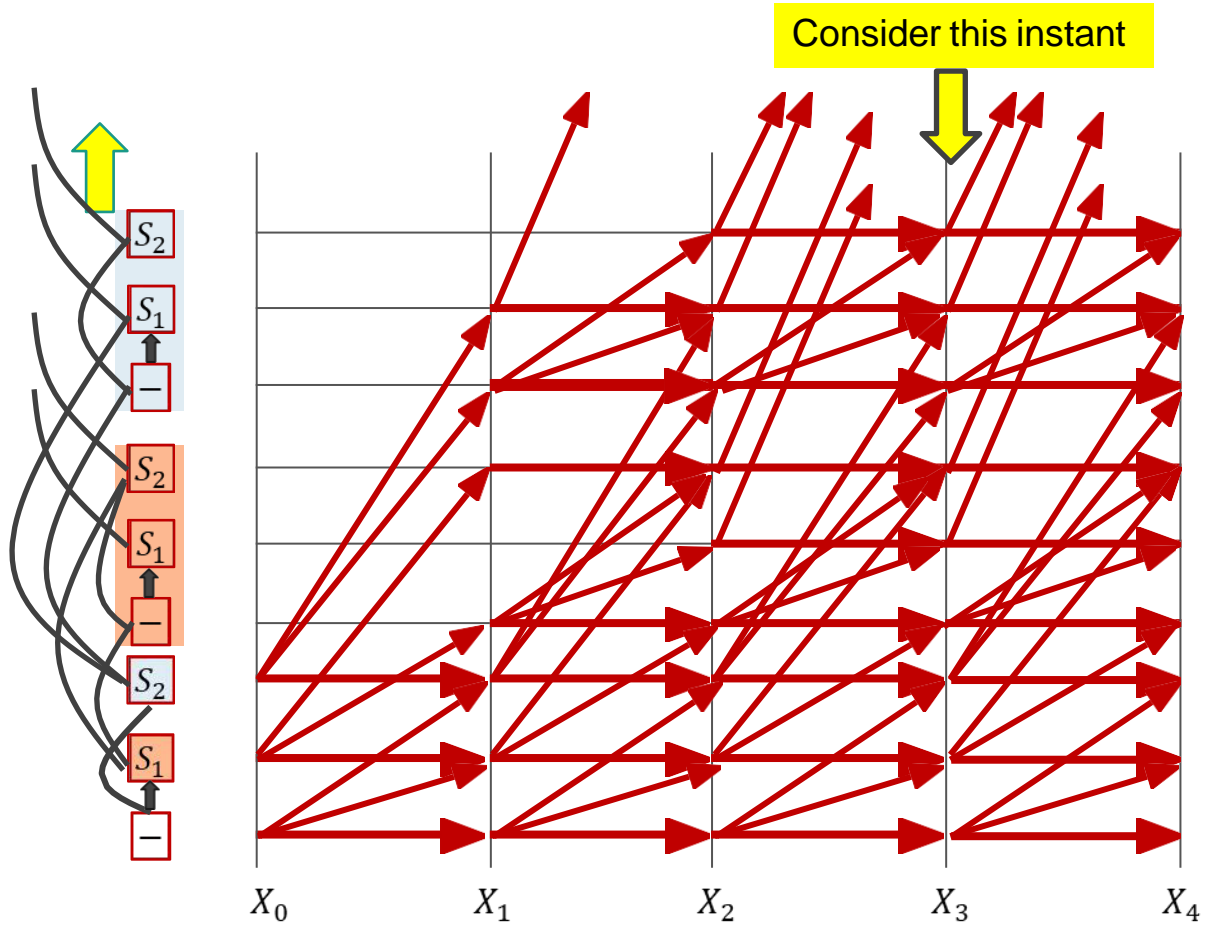
```
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore,
                                                    NewPathsWithTerminalSymbol, NewPathScore)
```

```
# Pick best path
```

```
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

Returning to this routine

Pruning deletes unpromising paths from contention, to reduce computation



# BEAM SEARCH: Pruning low-scoring entries

Global PathScore, BlankPathScore

```
function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
    PrunedBlankPathScore = []
    PrunedPathScore = []
    # First gather all the relevant scores
    i = 1
    for p in PathsWithTerminalBlank
        scorelist[i] = BlankPathScore[p]
        i++
    end
    for p in PathsWithTerminalSymbol
        scorelist[i] = PathScore[p]
        i++
    end

    # Sort and find cutoff score that retains exactly BeamWidth paths
    sort(scorelist) # In decreasing order
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]

    PrunedPathsWithTerminalBlank = {}
    for p in PathsWithTerminalBlank
        if BlankPathScore[p] >= cutoff
            PrunedPathsWithTerminalBlank += p # Set addition
            PrunedBlankPathScore[p] = BlankPathScore[p]
        end
    end

    PrunedPathsWithTerminalSymbol = {}
    for p in PathsWithTerminalSymbol
        if PathScore[p] >= cutoff
            PrunedPathsWithTerminalSymbol += p # Set addition
            PrunedPathScore[p] = PathScore[p]
        end
    end

    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
end
```



# BEAM SEARCH: Pruning low-scoring entries

Global PathScore, BlankPathScore

```
function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
```

```
    PrunedBlankPathScore = []
```

```
    PrunedPathScore = []
```

```
    # First gather all the relevant scores
```

```
    i = 1
```

```
    for p in PathsWithTerminalBlank
```

```
        scorelist[i] = BlankPathScore[p]
```

```
        i++
```

```
    end
```

```
    for p in PathsWithTerminalSymbol
```

```
        scorelist[i] = PathScore[p]
```

```
        i++
```

```
    end
```

```
    # Sort and find cutoff score that retains exactly BeamWidth paths
```

```
    sort(scorelist) # In decreasing order
```

```
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]
```

```
    PrunedPathsWithTerminalBlank = {}
```

```
    for p in PathsWithTerminalBlank
```

```
        if BlankPathScore[p] >= cutoff
```

```
            PrunedPathsWithTerminalBlank += # Set addition
```

```
            PrunedBlankPathScore[p] = BlankPathScore[p]
```

```
        end
```

```
    end
```

```
    PrunedPathsWithTerminalSymbol = {}
```

```
    for p in PathsWithTerminalSymbol
```

```
        if PathScore[p] >= cutoff
```

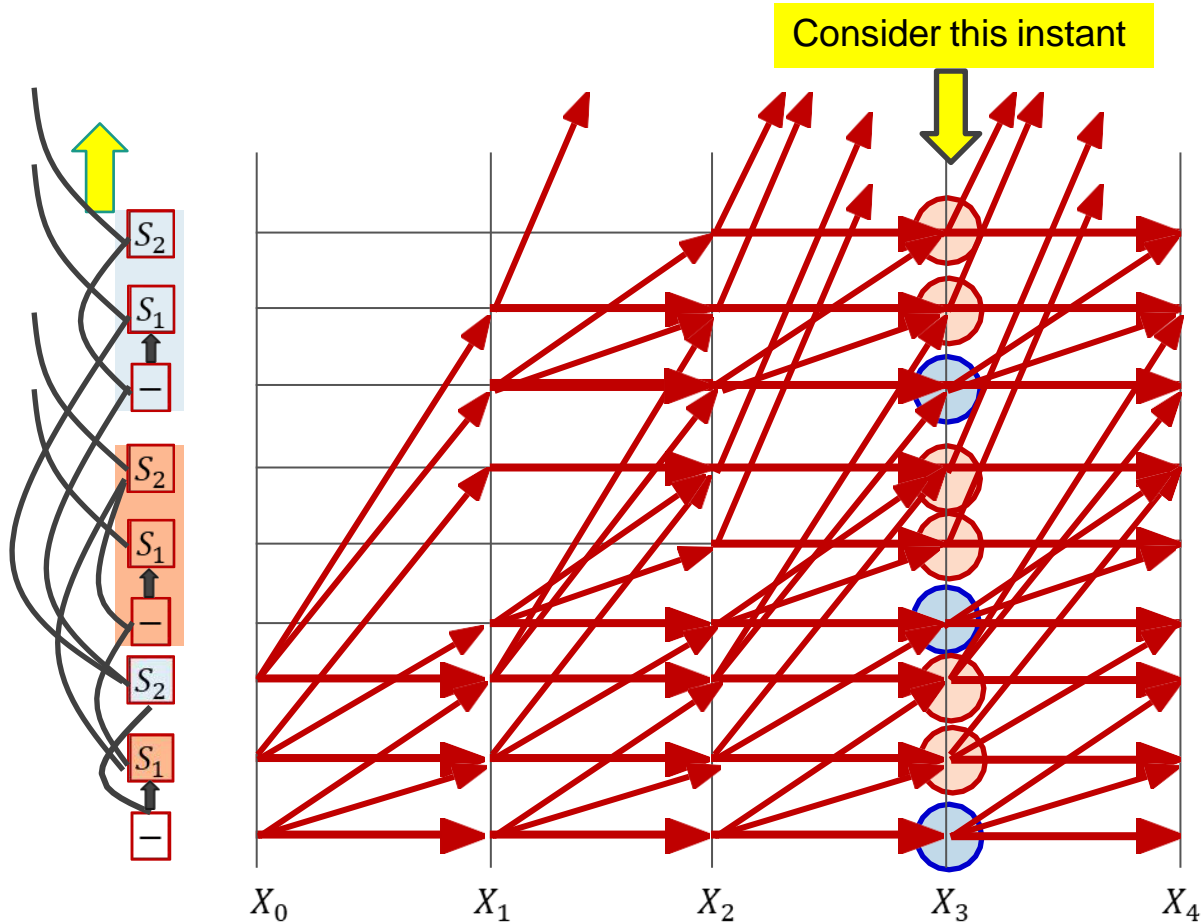
```
            PrunedPathsWithTerminalSymbol += # Set addition
```

```
            PrunedPathScore[p] = PathScore[p]
```

```
        end
```

```
    end
```

```
    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
```



# BEAM SEARCH: Pruning low-scoring entries

Global PathScore, BlankPathScore

```
function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
```

```
    PrunedBlankPathScore = []
```

```
    PrunedPathScore = []
```

```
    # First gather all the relevant scores
```

```
    i = 1
```

```
    for p in PathsWithTerminalBlank
```

```
        scorelist[i] = BlankPathScore[p]
```

```
        i++
```

```
    end
```

```
    for p in PathsWithTerminalSymbol
```

```
        scorelist[i] = PathScore[p]
```

```
        i++
```

```
    end
```

```
    # Sort and find cutoff score that retains exactly BeamWidth paths
```

```
    sort(scorelist) # In decreasing order
```

```
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]
```

```
    PrunedPathsWithTerminalBlank = {}
```

```
    for p in PathsWithTerminalBlank
```

```
        if BlankPathScore[p] >= cutoff
```

```
            PrunedPathsWithTerminalBlank += p # Set addition
```

```
            PrunedBlankPathScore[p] = BlankPathScore[p]
```

```
        end
```

```
    end
```

```
    PrunedPathsWithTerminalSymbol = {}
```

```
    for p in PathsWithTerminalSymbol
```

```
        if PathScore[p] >= cutoff
```

```
            PrunedPathsWithTerminalSymbol += p # Set addition
```

```
            PrunedPathScore[p] = PathScore[p]
```

```
        end
```

```
    end
```

```
    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
```

Sort the scores

Find the largest score

Find the cutoff score (the Kth largest score)

# BEAM SEARCH: Pruning low-scoring entries

Global PathScore, BlankPathScore

```
function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
    PrunedBlankPathScore = []
    PrunedPathScore = []
    # First gather all the relevant scores
    i = 1
    for p in PathsWithTerminalBlank
        scorelist[i] = BlankPathScore[p]
        i++
    end
    for p in PathsWithTerminalSymbol
        scorelist[i] = PathScore[p]
        i++
    end

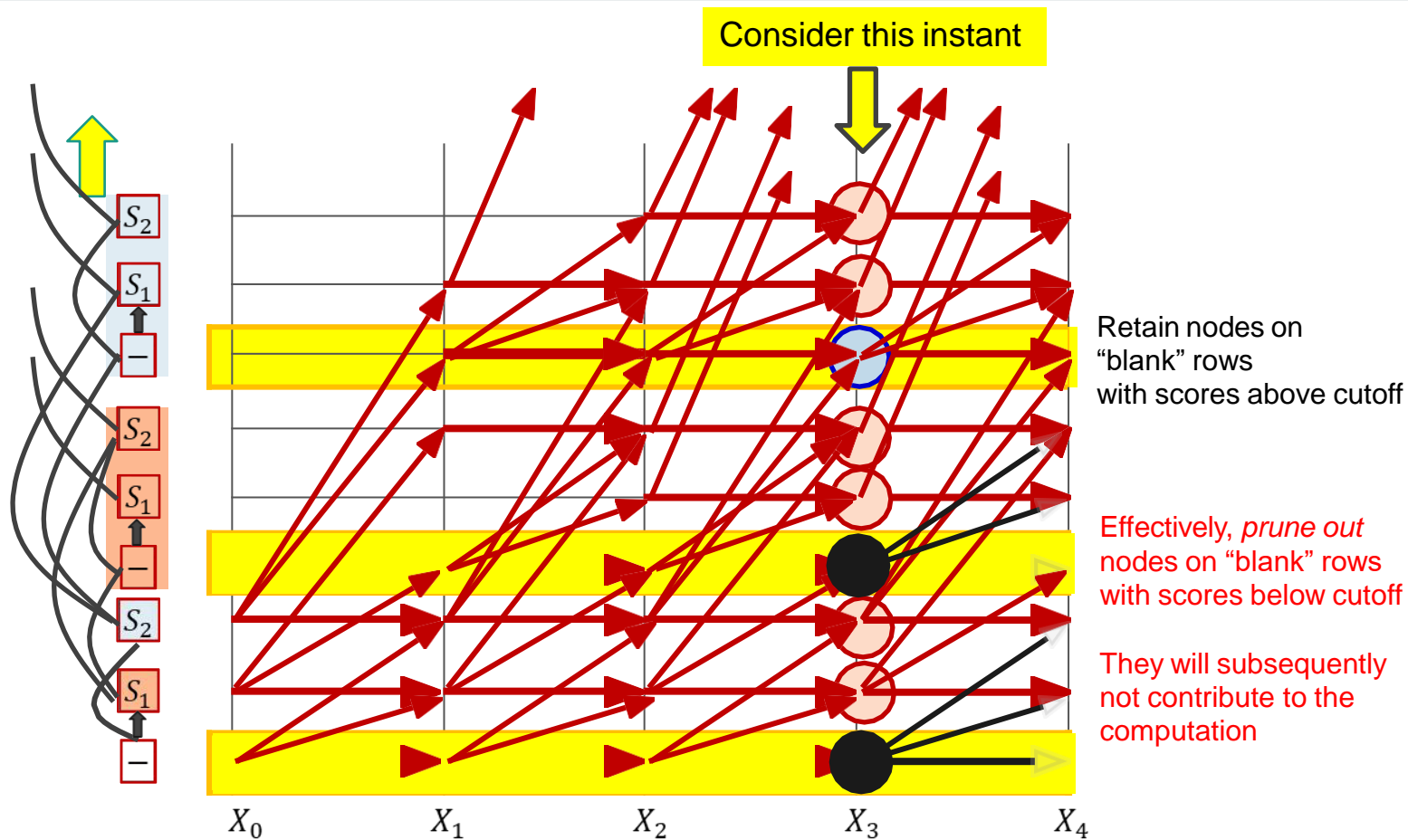
    # Sort and find cutoff score that retains exactly BeamWidth paths
    sort(scorelist) # In decreasing order
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]

    PrunedPathsWithTerminalBlank = {}
    for p in PathsWithTerminalBlank
        if BlankPathScore[p] >= cutoff
            PrunedPathsWithTerminalBlank += p # Set addition
            PrunedBlankPathScore[p] = BlankPathScore[p]
        end
    end

    PrunedPathsWithTerminalSymbol = {}
    for p in PathsWithTerminalSymbol
        if PathScore[p] >= cutoff
            PrunedPathsWithTerminalSymbol += p # Set addition
            PrunedPathScore[p] = PathScore[p]
        end
    end

    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
end
```

Find nodes on  
“blank” rows  
with scores above cutoff  
and add them to the  
“active” list



# BEAM SEARCH: Pruning low-scoring entries

Global PathScore, BlankPathScore

```
function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
    PrunedBlankPathScore = []
    PrunedPathScore = []
    # First gather all the relevant scores
    i = 1
    for p in PathsWithTerminalBlank
        scorelist[i] = BlankPathScore[p]
        i++
    end
    for p in PathsWithTerminalSymbol
        scorelist[i] = PathScore[p]
        i++
    end

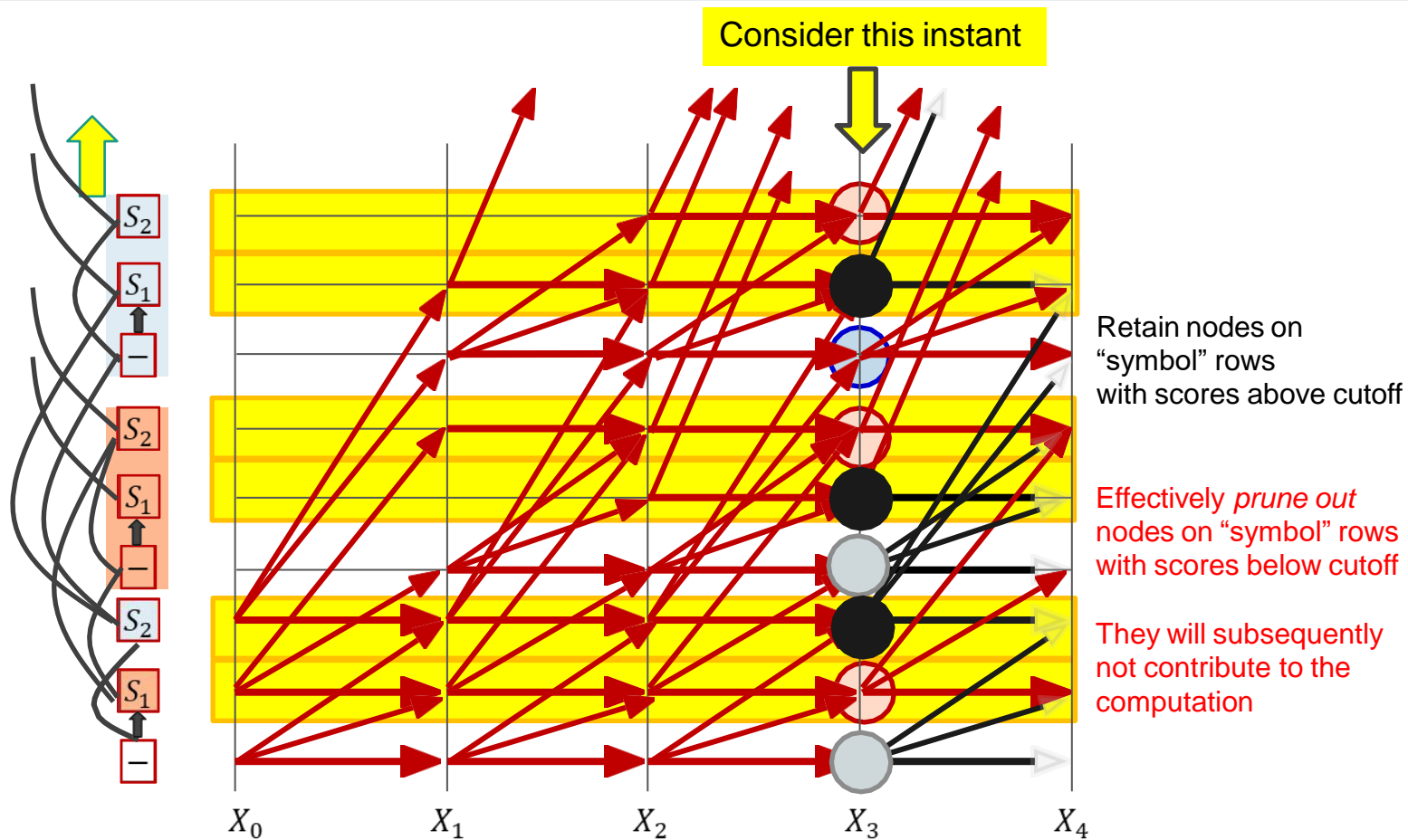
    # Sort and find cutoff score that retains exactly BeamWidth paths
    sort(scorelist) # In decreasing order
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]

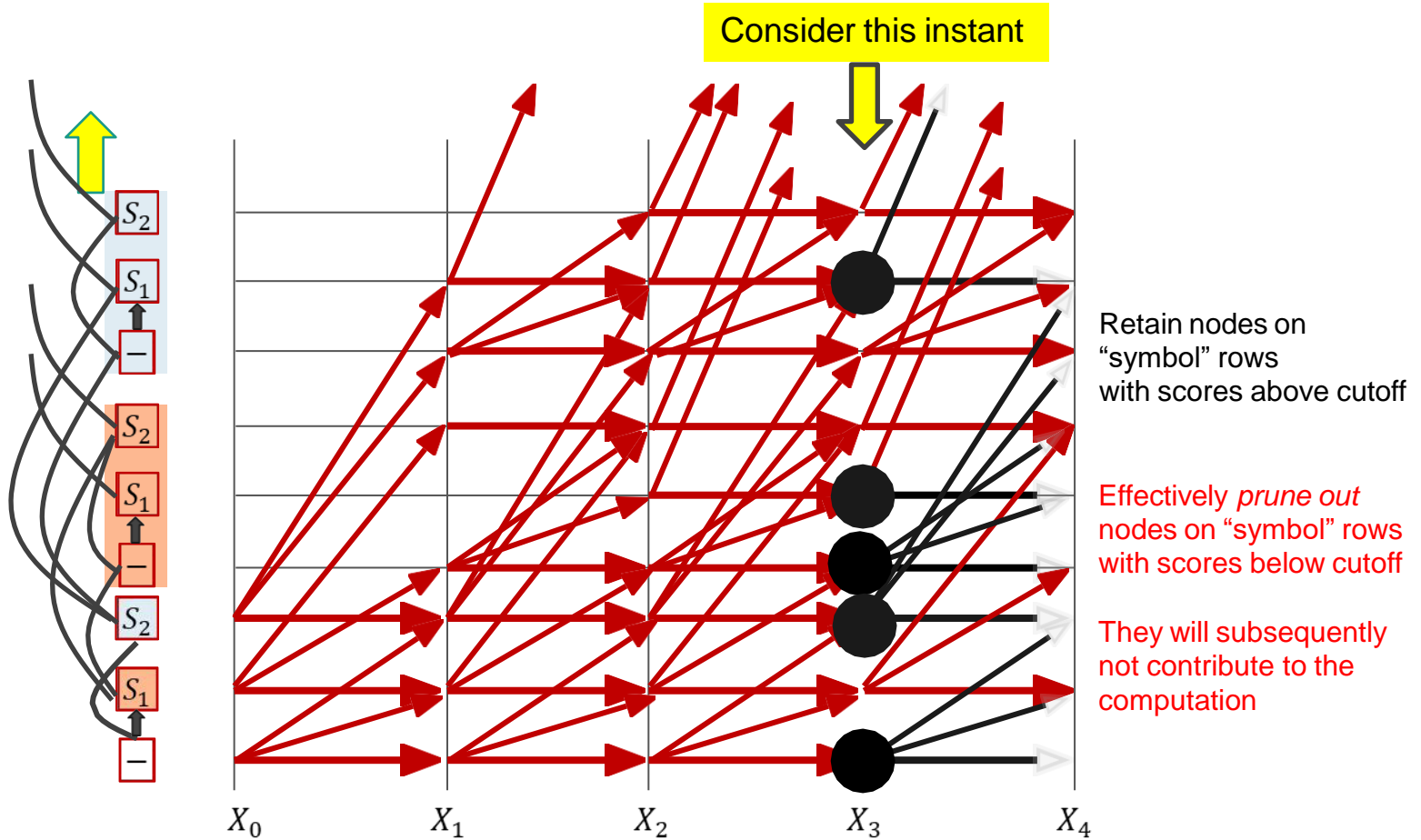
    PrunedPathsWithTerminalBlank = {}
    for p in PathsWithTerminalBlank
        if BlankPathScore[p] >= cutoff
            PrunedPathsWithTerminalBlank += p # Set addition
            PrunedBlankPathScore[p] = BlankPathScore[p]
        end
    end

    PrunedPathsWithTerminalSymbol = {}
    for p in PathsWithTerminalSymbol
        if PathScore[p] >= cutoff
            PrunedPathsWithTerminalSymbol += p # Set addition
            PrunedPathScore[p] = PathScore[p]
        end
    end

    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
end
```

Find nodes on  
“symbol” rows  
with scores above cutoff  
and add them to the  
“active” list







# BEAM SEARCH: Pruning low-scoring entries

Global PathScore, BlankPathScore

```
function Prune(PathsWithTerminalBlank, PathsWithTerminalSymbol, BlankPathScore, PathScore, BeamWidth)
```

```
    PrunedBlankPathScore = []
```

```
    PrunedPathScore = []
```

```
    # First gather all the relevant scores
```

```
    i = 1
```

```
    for p in PathsWithTerminalBlank
```

```
        scorelist[i] = BlankPathScore[p]
```

```
        i++
```

```
    end
```

```
    for p in PathsWithTerminalSymbol
```

```
        scorelist[i] = PathScore[p]
```

```
        i++
```

```
    end
```

```
    # Sort and find cutoff score that retains exactly BeamWidth paths
```

```
    sort(scorelist) # In decreasing order
```

```
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]
```

```
    PrunedPathsWithTerminalBlank = {}
```

```
    for p in PathsWithTerminalBlank
```

```
        if BlankPathScore[p] >= cutoff
```

```
            PrunedPathsWithTerminalBlank += # Set addition
```

```
            PrunedBlankPathScore[p] = BlankPathScore[p]
```

```
        end
```

```
    end
```

```
    PrunedPathsWithTerminalSymbol = {}
```

```
    for p in PathsWithTerminalSymbol
```

```
        if PathScore[p] >= cutoff
```

```
            PrunedPathsWithTerminalSymbol += # Set addition
```

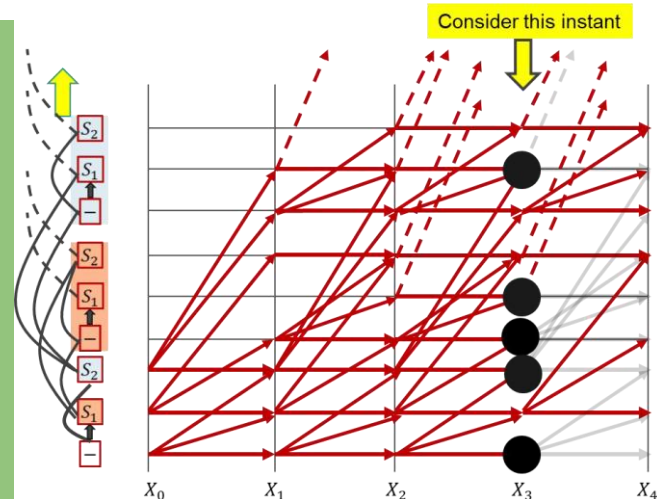
```
            PrunedPathScore[p] = PathScore[p]
```

```
        end
```

```
    end
```

```
    return PrunedPathsWithTerminalBlank, PrunedPathsWithTerminalSymbol, PrunedBlankPathScore, PrunedPathScore
```

The overall effect of these steps:



# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)

    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(
        PathsWithTerminalBlank, BlankPathScore)

    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(
        PathsWithTerminalBlank, BlankPathScore, PathsWithTerminalSymbol,
        PathScore)

end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore,
                                                    NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```

Why is the pruning here and not at the end of the loop?

Because we don't want to prune paths at the final time. This loses information.

Instead at the final time we will merge paths that represent the same symbol sequence

# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)

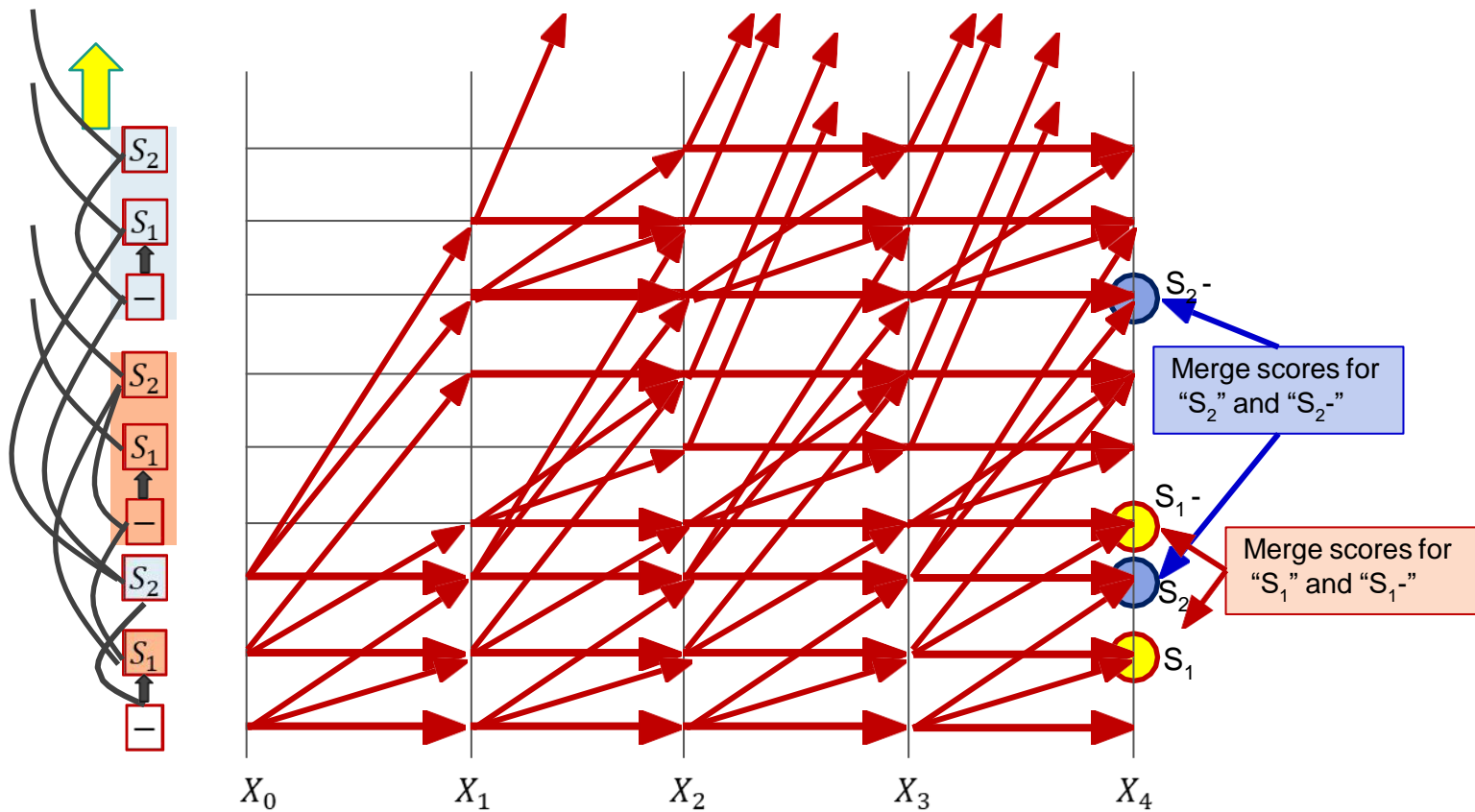
    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])

end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore
                                                  NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```



## BEAM SEARCH: Merging final paths

Global PathScore, BlankPathScore

```
function MergeIdenticalPaths (PathsWithTerminalBlank, PathsWithTerminalSymbol)

    # All paths with terminal symbols will remain
    MergedPaths = PathsWithTerminalSymbol
    FinalPathScore = PathScore

    # Paths with terminal blanks will contribute scores to existing identical paths from
    # PathsWithTerminalSymbol if present, or be included in the final set, otherwise
    for p in PathsWithTerminalBlank
        if p in MergedPaths
            FinalPathScore[p] += BlankPathScore[p]
        else
            MergedPaths += p # Set addition
            FinalPathScore[p] = BlankPathScore[p]
        end
    end

    return MergedPaths, FinalPathScore
```

# BEAM SEARCH

```
Global PathScore = [], BlankPathScore = []

# First time instant: Initialize paths with each of the symbols,
# including blank, using score at time t=1
NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol, NewBlankPathScore, NewPathScore =
    InitializePaths(SymbolSet, y[:,0])

# Subsequent time steps
for t = 1:T
    # Prune the collection down to the BeamWidth
    PathsWithTerminalBlank, PathsWithTerminalSymbol, PathScore, BlankPathScore =
        Prune(NewPathsWithTerminalBlank, NewPathsWithTerminalSymbol,
              NewBlankPathScore, NewPathScore, BeamWidth)

    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])

end

# Merge identical paths differing only by the final blank
MergedPaths, FinalPathScore = MergeIdenticalPaths(NewPathsWithTerminalBlank, NewBlankPathScore
                                                    NewPathsWithTerminalSymbol, NewPathScore)

# Pick best path
BestPath = argmax(FinalPathScore) # Find the path with the best score
```