



# Recitation 8

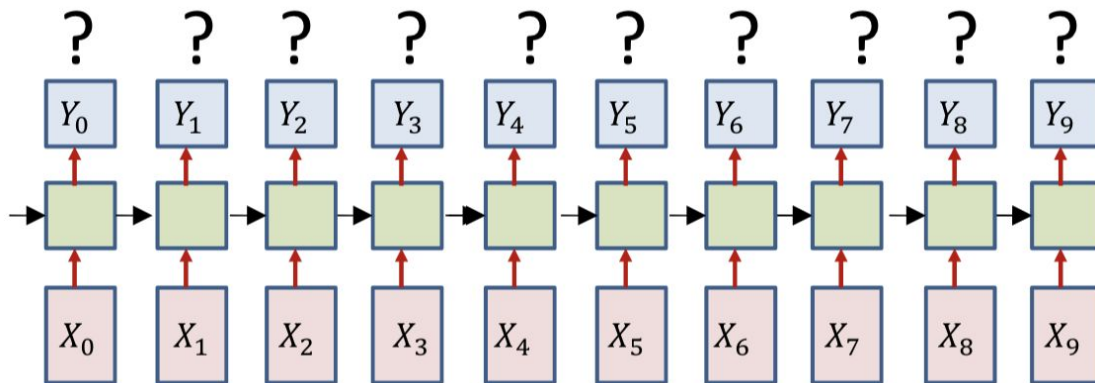
## CTC Decoding & Beam Search

Soumya Vadlamannati, Amala Deshmukh

# Sequence to Sequence Modeling

Order-Synchronous, Not Time-Synchronous Output

1. Training -> we already know how to do that
2. Testing -> “Decoding” or “obtaining an output from a sequence-to-sequence network”





# A key decoding problem

- Consider a problem where the output symbols are characters
- We have a decode: RRROOOOOD
- Is this the merged symbol sequence ROD or ROOD?

How to distinguish between an extended symbol and repetitions of a symbol?



# A key decoding problem

Solution: Introduce an explicit extra symbol which serves to separate discrete versions of a symbol

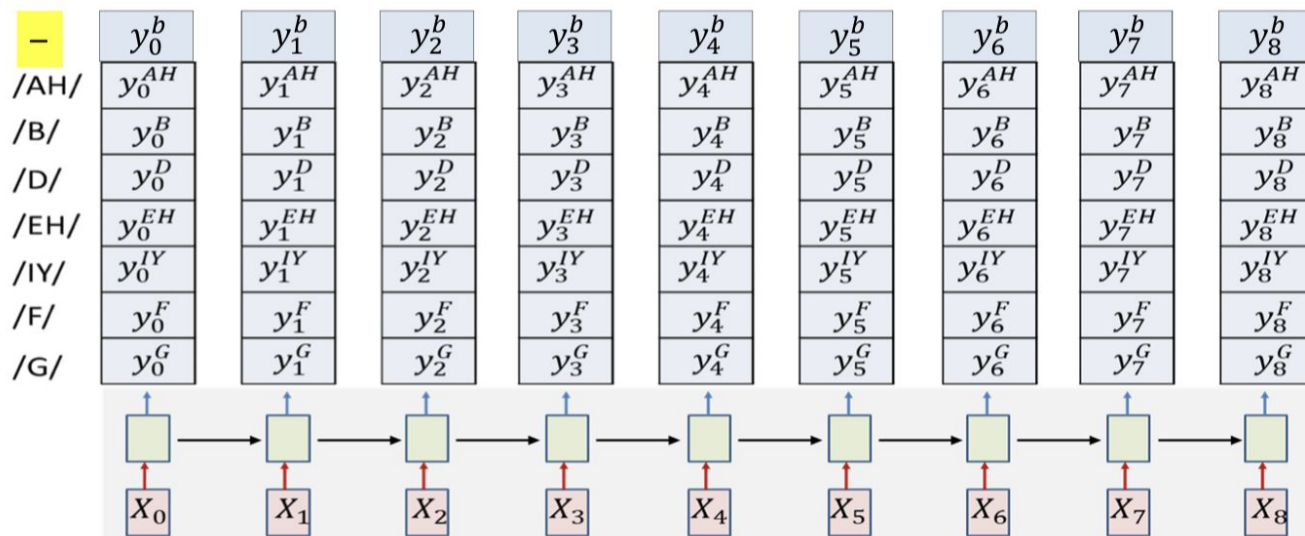
- A “blank” (represented by “-”)
- RRR---OO---DDD = ROD
- RR-R---OO---D-DD = RRODD
- R-R-R---O-ODD-DDDD-D = RRROODDD

The symbol set recognized by the network must now include the extra blank symbol

- Which too must be trained

# The modified forward output

Note the extra “blank” at the output



# Composing graph for training

—	$y_0^b$	$y_1^b$	$y_2^b$	$y_3^b$	$y_4^b$	$y_5^b$	$y_6^b$	$y_7^b$	$y_8^b$
/B/	$y_0^B$	$y_1^B$	$y_2^B$	$y_3^B$	$y_4^B$	$y_5^B$	$y_6^B$	$y_7^B$	$y_8^B$
—	$y_0^b$	$y_1^b$	$y_2^b$	$y_3^b$	$y_4^b$	$y_5^b$	$y_6^b$	$y_7^b$	$y_8^b$
/IY/	$y_0^{IY}$	$y_1^{IY}$	$y_2^{IY}$	$y_3^{IY}$	$y_4^{IY}$	$y_5^{IY}$	$y_6^{IY}$	$y_7^{IY}$	$y_8^{IY}$
—	$y_0^b$	$y_1^b$	$y_2^b$	$y_3^b$	$y_4^b$	$y_5^b$	$y_6^b$	$y_7^b$	$y_8^b$
/IY/	$y_0^{IY}$	$y_1^{IY}$	$y_2^{IY}$	$y_3^{IY}$	$y_4^{IY}$	$y_5^{IY}$	$y_6^{IY}$	$y_7^{IY}$	$y_8^{IY}$
—	$y_0^b$	$y_1^b$	$y_2^b$	$y_3^b$	$y_4^b$	$y_5^b$	$y_6^b$	$y_7^b$	$y_8^b$
/F/	$y_0^F$	$y_1^F$	$y_2^F$	$y_3^F$	$y_4^F$	$y_5^F$	$y_6^F$	$y_7^F$	$y_8^F$
—	$y_0^b$	$y_1^b$	$y_2^b$	$y_3^b$	$y_4^b$	$y_5^b$	$y_6^b$	$y_7^b$	$y_8^b$

Train as  
before!

- With blanks
- Note: a row of blanks between any two symbols
- Also blanks at the very beginning and the very end



# CTC: Connectionist Temporal Classification

- The overall framework we saw is referred to as CTC
  - Applies when “duplicating” labels at the output is considered acceptable, and when output sequence length < input sequence length



# Returning to the decoding problem

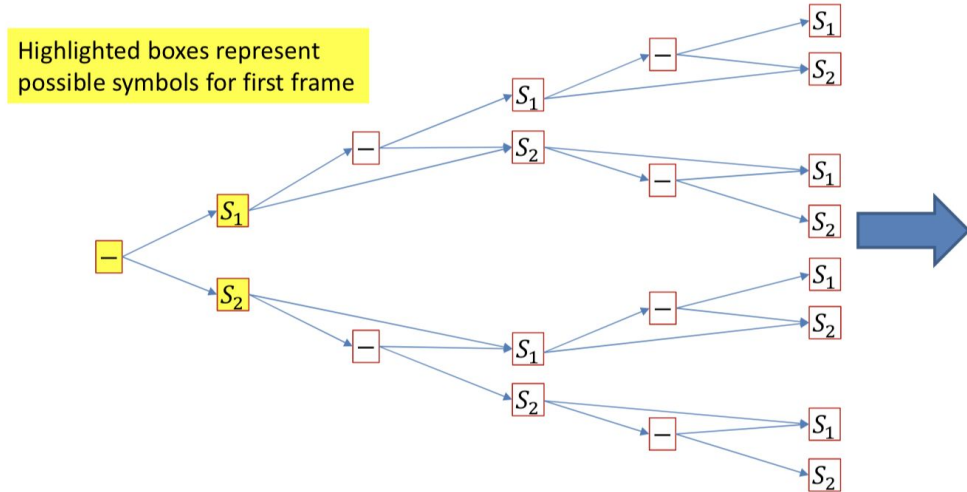
How to decode at test time?

- Greedy decode -> choose symbol with highest probability at each time step and merge
  - Sub-optimal decode which finds most likely synchronous output sequence
- Objective of decoding -> Most likely asynchronous symbol sequence
  - Find all decodings and pick the most likely decode!
  - Unfortunately, explicit computation of this will require evaluate of an exponential number of symbol sequences
  - Solution: Organize all possible symbol sequences as a (semi)tree



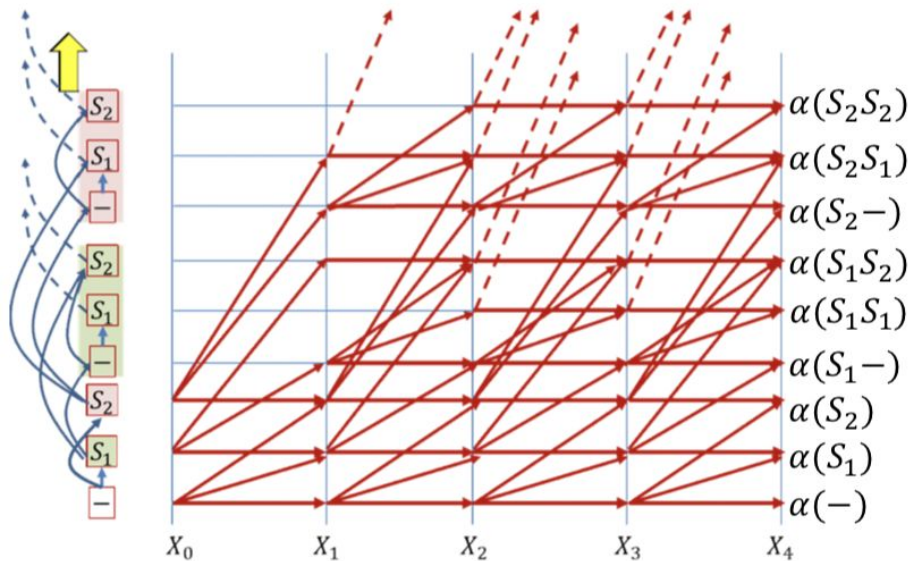
# Hypothesis semi-tree

- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
- It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique symbol sequence!



# Decoding graph for the tree

- The figure to the left is the tree, drawn in a vertical line
- The graph is just the tree unrolled over time
- The alpha at final time represents the full forward score for a unique symbol sequence
- Select the symbol sequence with the largest alpha





# Pruning

- This is the “theoretically correct” CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable

## Beam Search

- **PathScore** : array of scores for paths ending with symbols
- **BlankPathScore** : array of scores for paths ending with blanks
- **SymbolSet** : A list of symbols *not* including the blank

## 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])
```

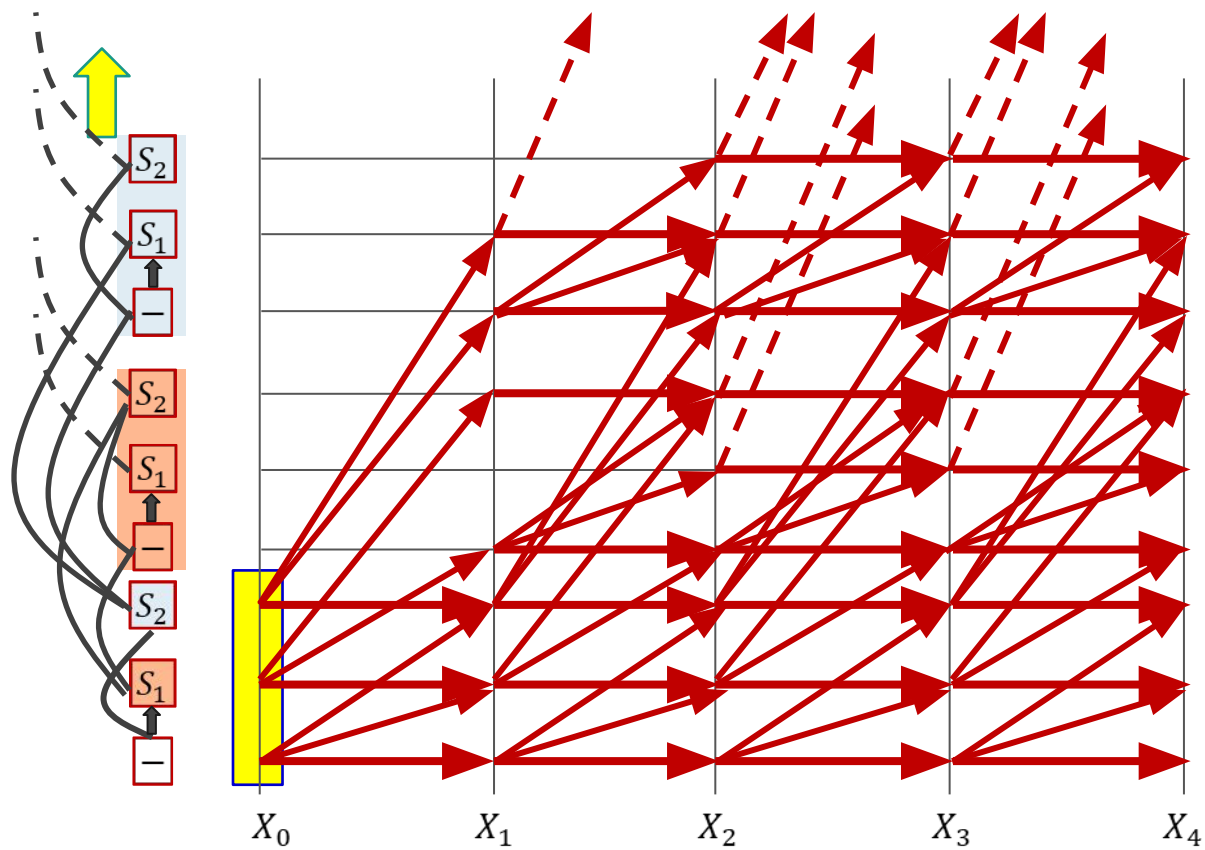
```
end
```

```
# 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

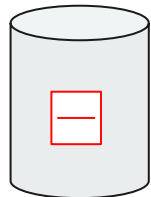
```
function InitializePaths(SymbolSet, y)

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}

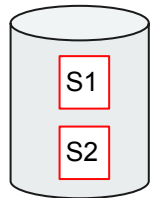
# Push rest of the symbols into a path-ending-with-symbol stack
InitialPathsWithFinalSymbol = {}
for c in SymbolSet # This is the entire symbol set, without the blank
    path = c
    InitialPathScore[path] = y[c] # Score of symbol c at t=1
    InitialPathsWithFinalSymbol += path # Set addition
end

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)

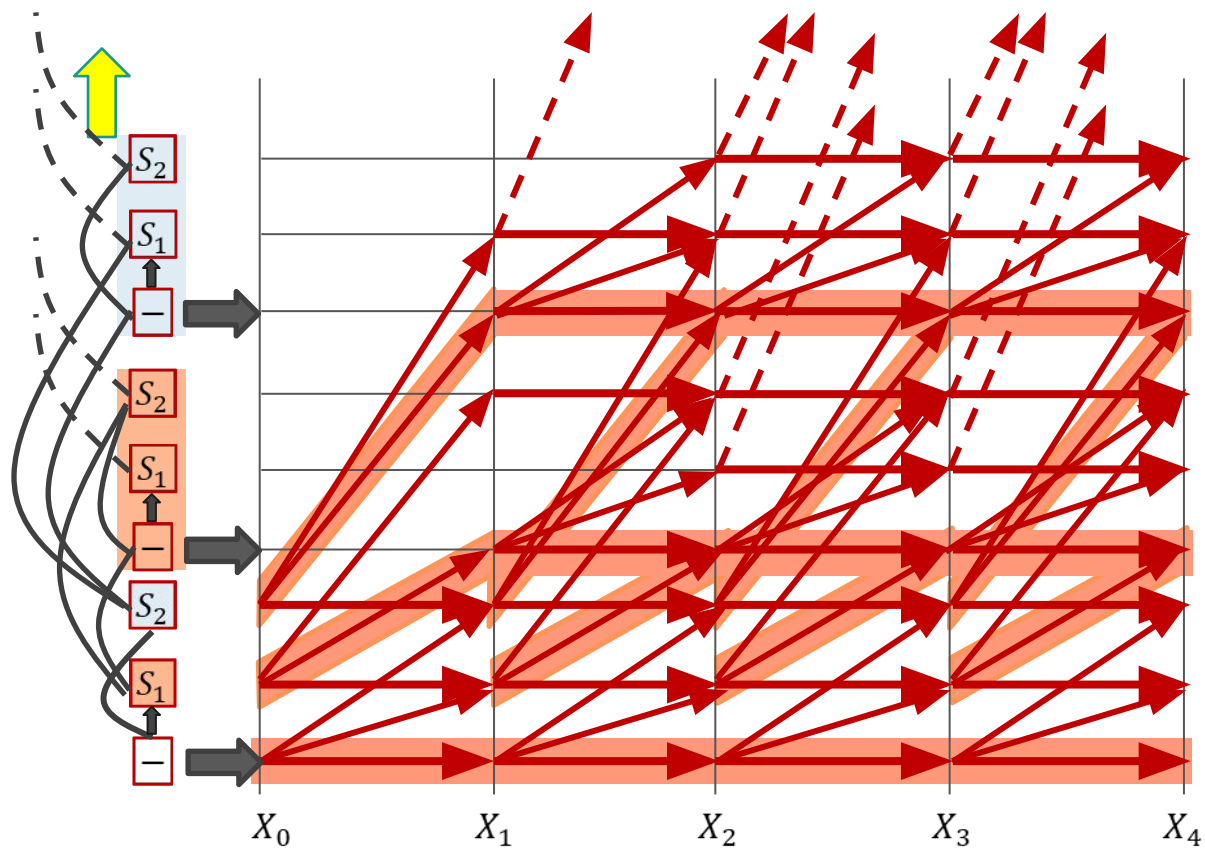
    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

    # 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
```



# BEAM SEARCH: Extending with blanks

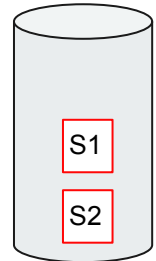
Global PathScore, BlankPathScore

```
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

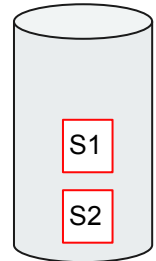
Global PathScore, BlankPathScore

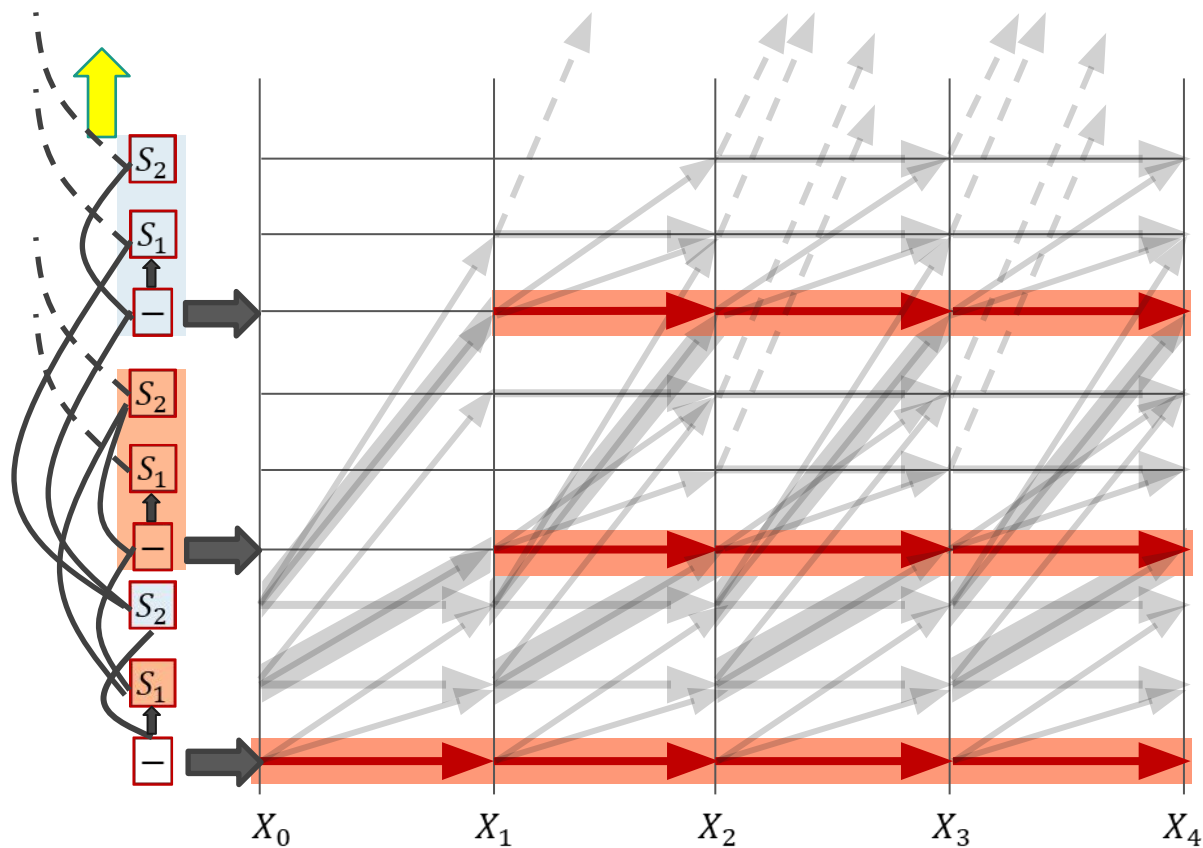
```
function ExtendWithBlank(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalBlank = {}
    UpdatedBlankPathScore = []
    # First work on paths with terminal blanks
    #(This represents transitions along horizontal trellis edges for blanks)
    for path in PathsWithTerminalBlank:
        # 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

    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

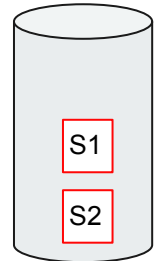
Global PathScore, BlankPathScore

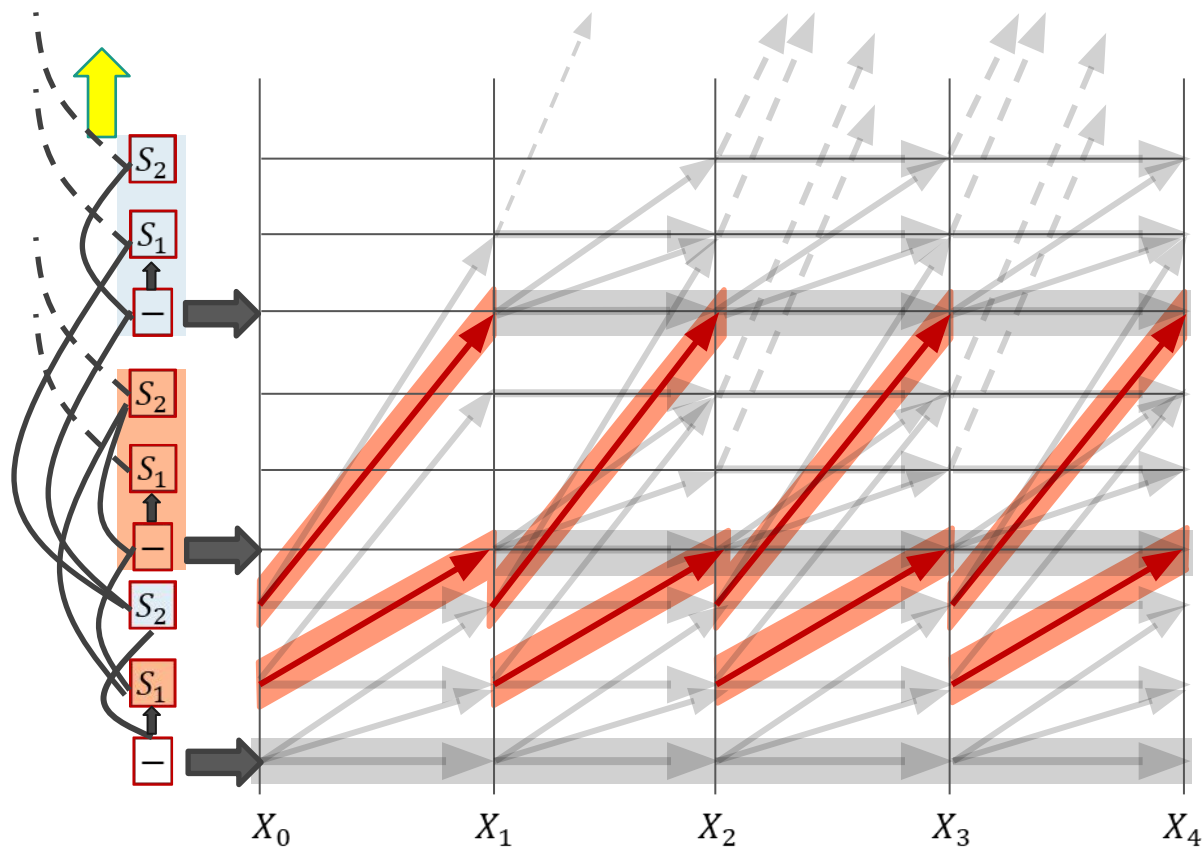
```
function ExtendWithBlank(PathsWithTerminalBlank, PathsWithTerminalSymbol, y)
    UpdatedPathsWithTerminalBlank = {}
    UpdatedBlankPathScore = []
    # First work on paths with terminal blanks
    #(This represents transitions along horizontal trellis edges for blanks)
    for path in PathsWithTerminalBlank:
        # 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
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])

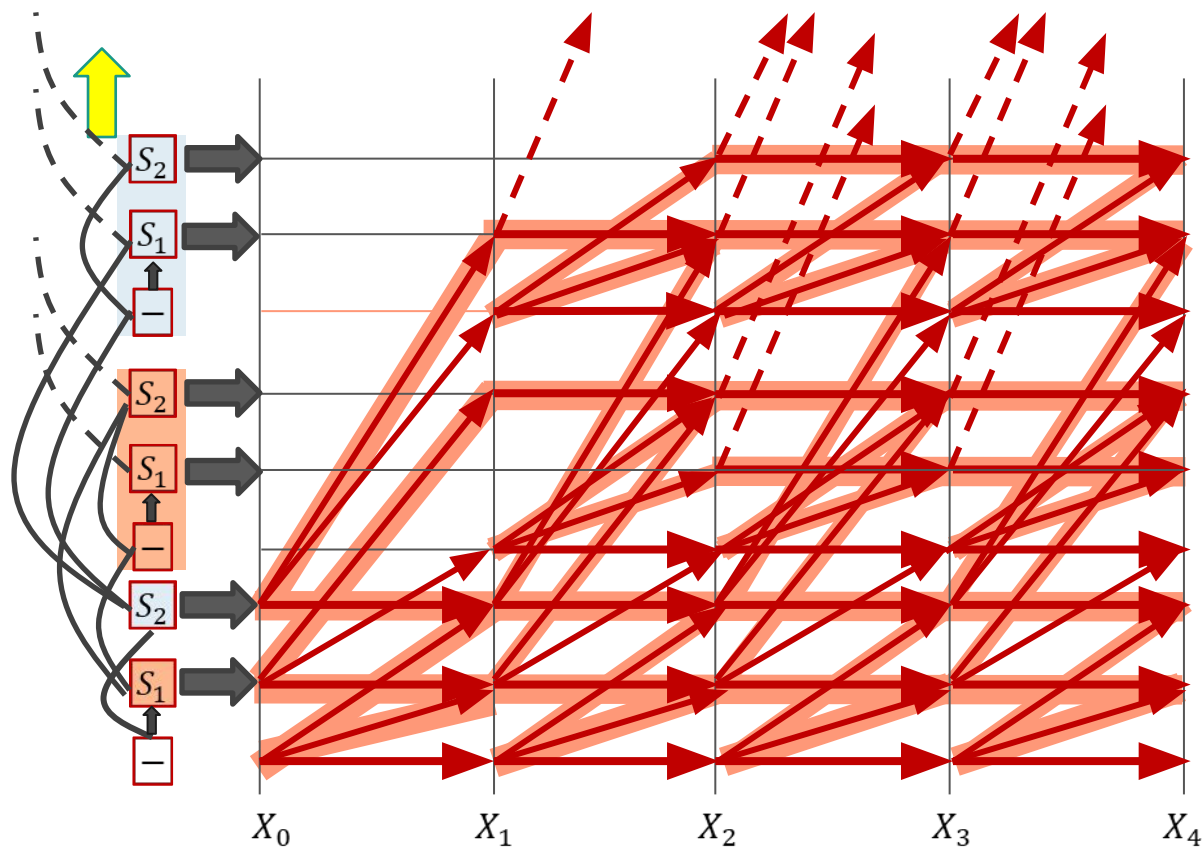
    # Next extend paths by a symbol
    NewPathsWithTerminalSymbol, NewPathScore = ExtendWithSymbol(PathsWithTerminalBlank,
                                                                PathsWithTerminalSymbol, SymbolSet, y[:,t])

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
```





(figure shows path extensions for only 2 time steps)

## BEAM SEARCH: Extending with symbols

Global PathScore, BlankPathScore

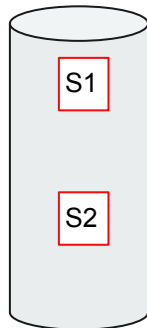
```
function ExtendWithSymbol(PathsWithTerminalBlank, PathsWithTerminalSymbol, SymbolSet, y)
    UpdatedPathsWithTerminalSymbol = {}
    UpdatedPathScore = []

    # 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

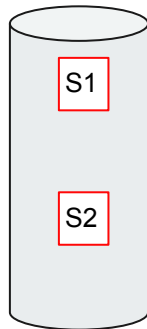
```
function ExtendWithSymbol(PathsWithTerminalBlank, PathsWithTerminalSymbol, SymbolSet, y)
    UpdatedPathsWithTerminalSymbol = {}
    UpdatedPathScore = []

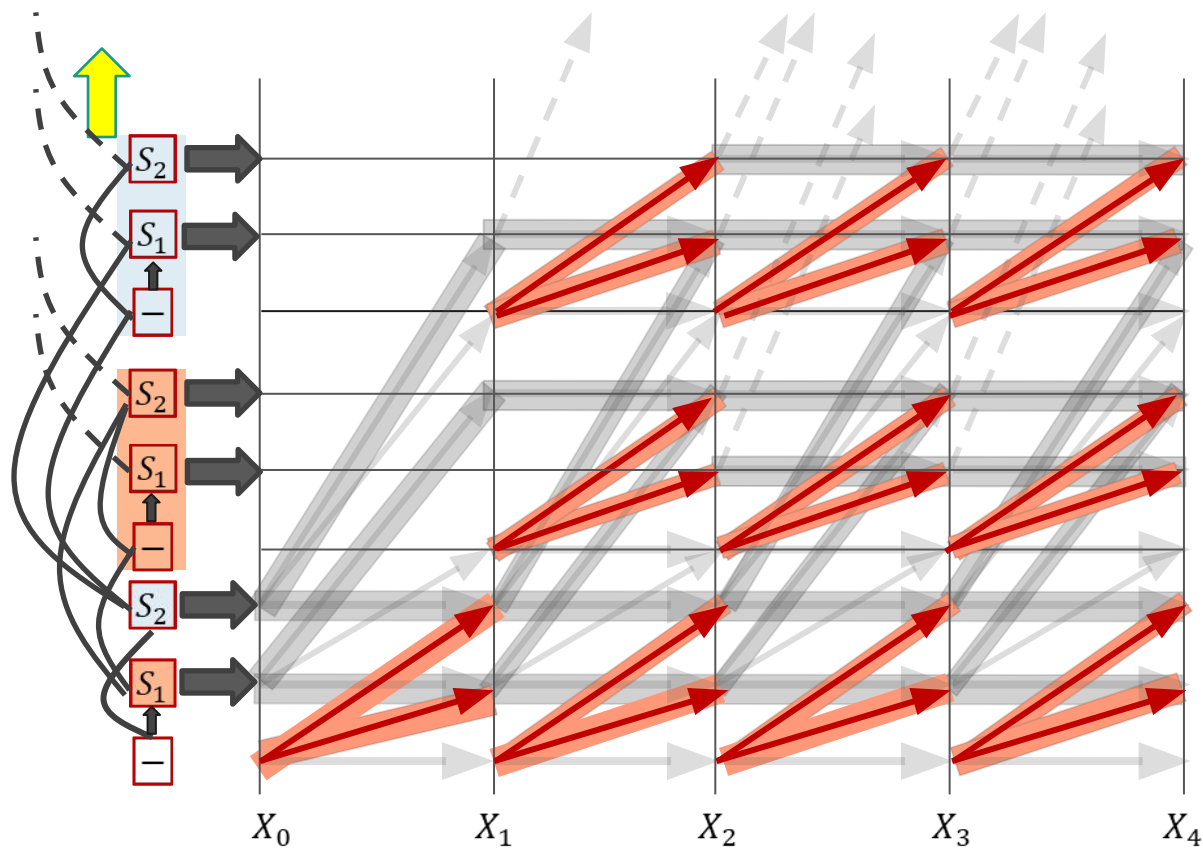
    # First extend the paths terminating in blanks. This will always create a new sequence
    for path in PathsWithTerminalBlank:
        for c in SymbolSet: # SymbolSet does not include blanks
            newpath = path + c # Concatenation
            UpdatedPathsWithTerminalSymbol += newpath # Set addition
            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]
```

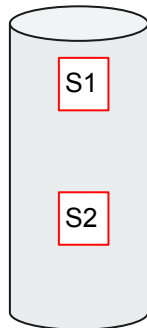
```
            end
```

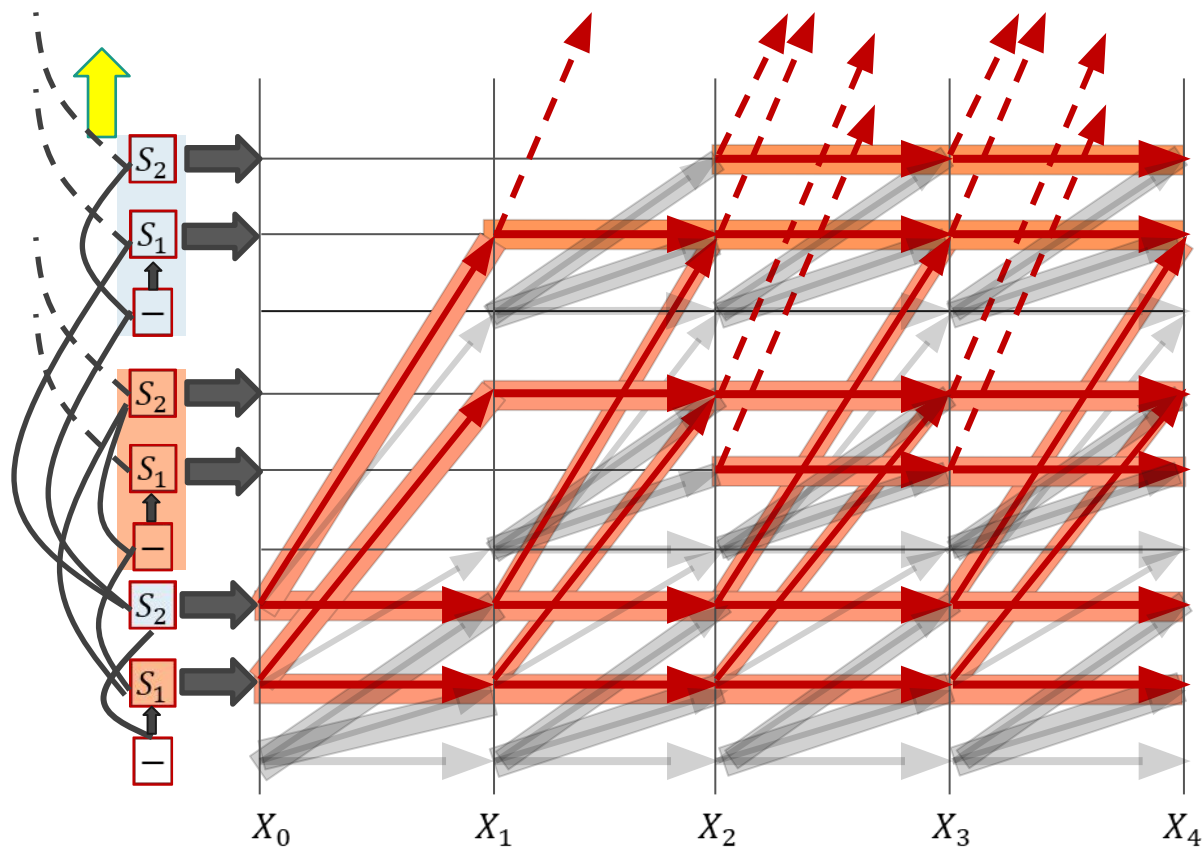
```
        end
```

```
    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(Paths
                                                                    Pat

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

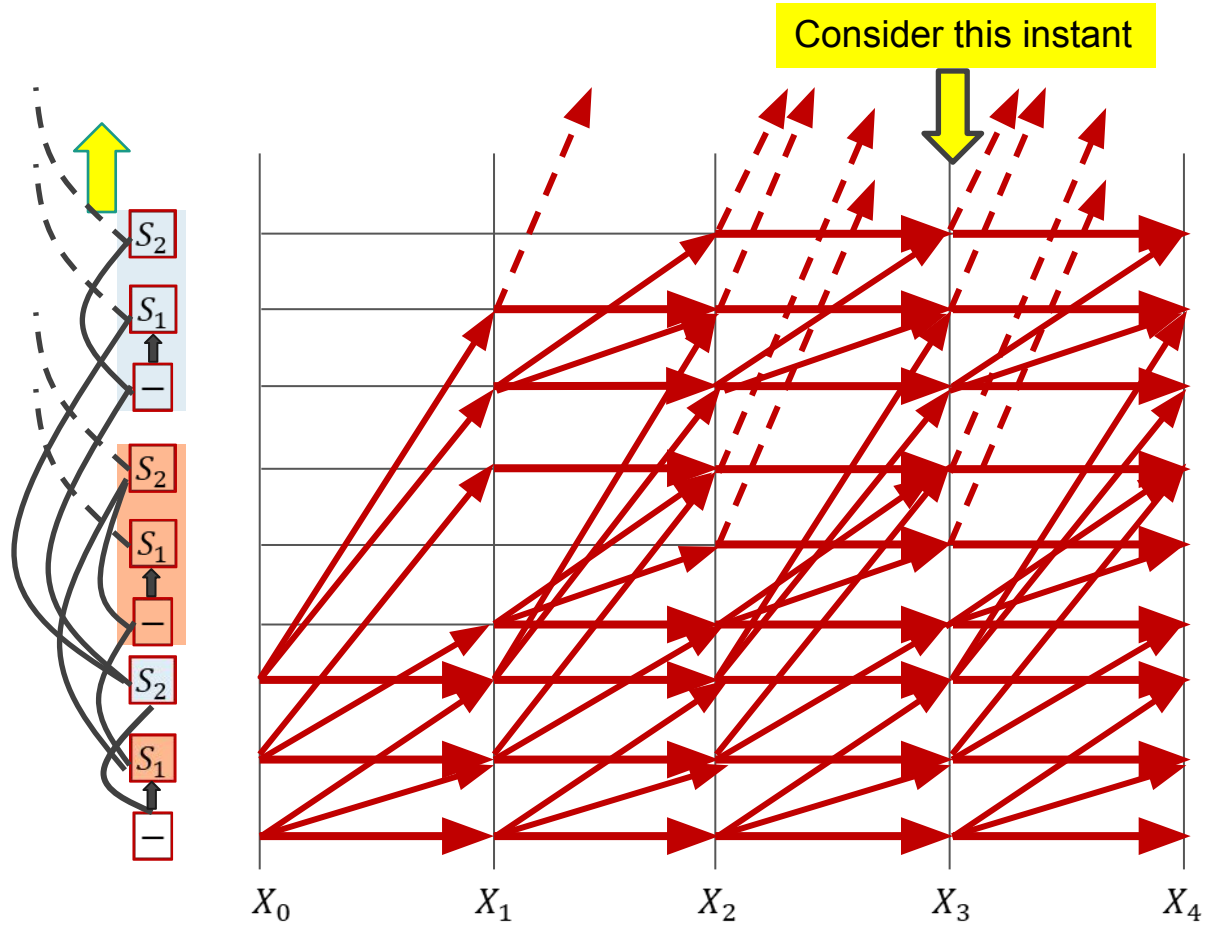
end

# 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 += p # Set addition
```

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

```
        end
```

```
    end
```

```
    PrunedPathsWithTerminalSymbol = {}
```

```
    for p in PathsWithTerminalSymbol
```

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

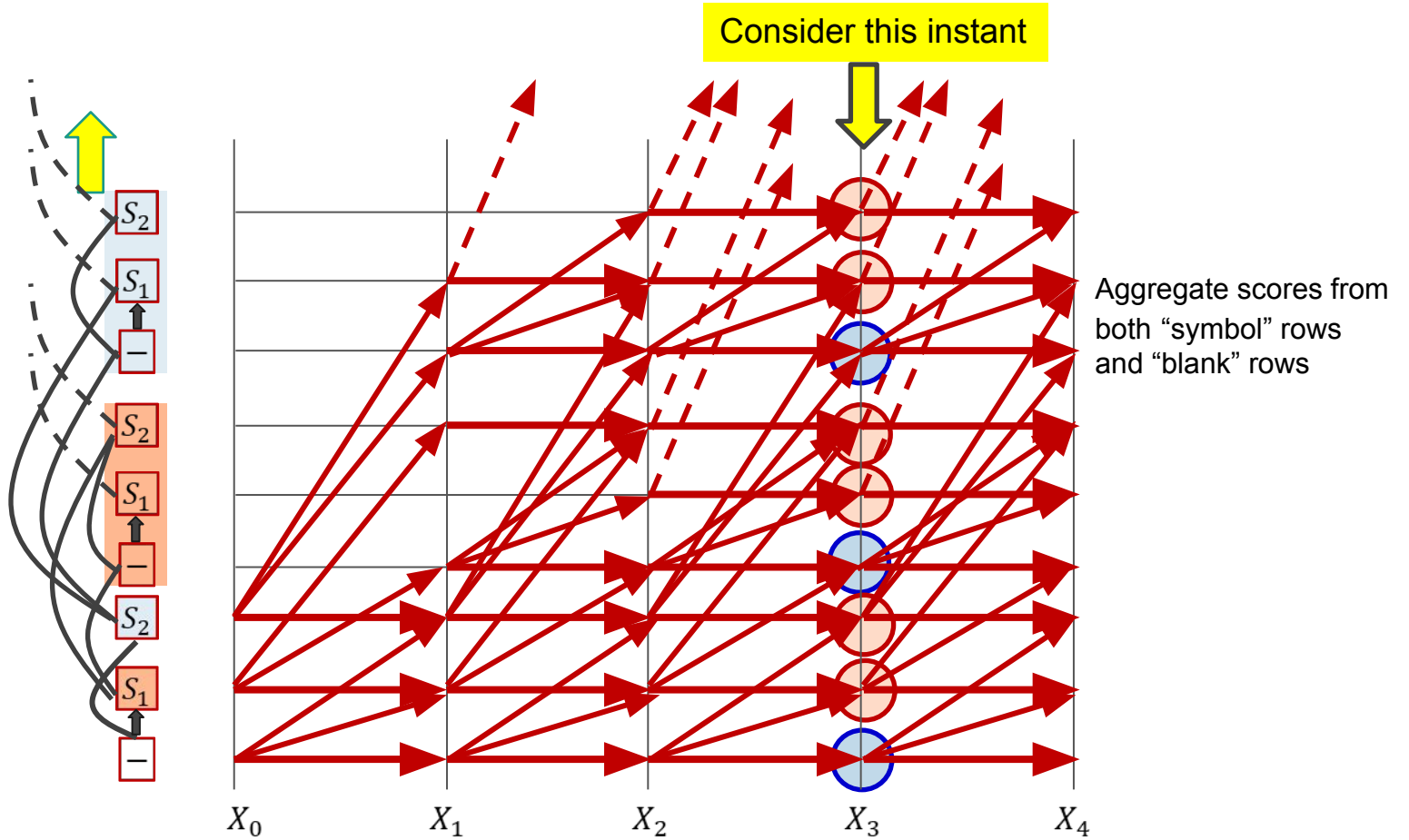
```
            PrunedPathsWithTerminalSymbol += p # Set addition
```

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

```
        end
```

```
    end
```

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



# 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 the scores

Find the largest score

Find the cutoff score (the Kth largest score)

```
    # 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
```

# 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
```

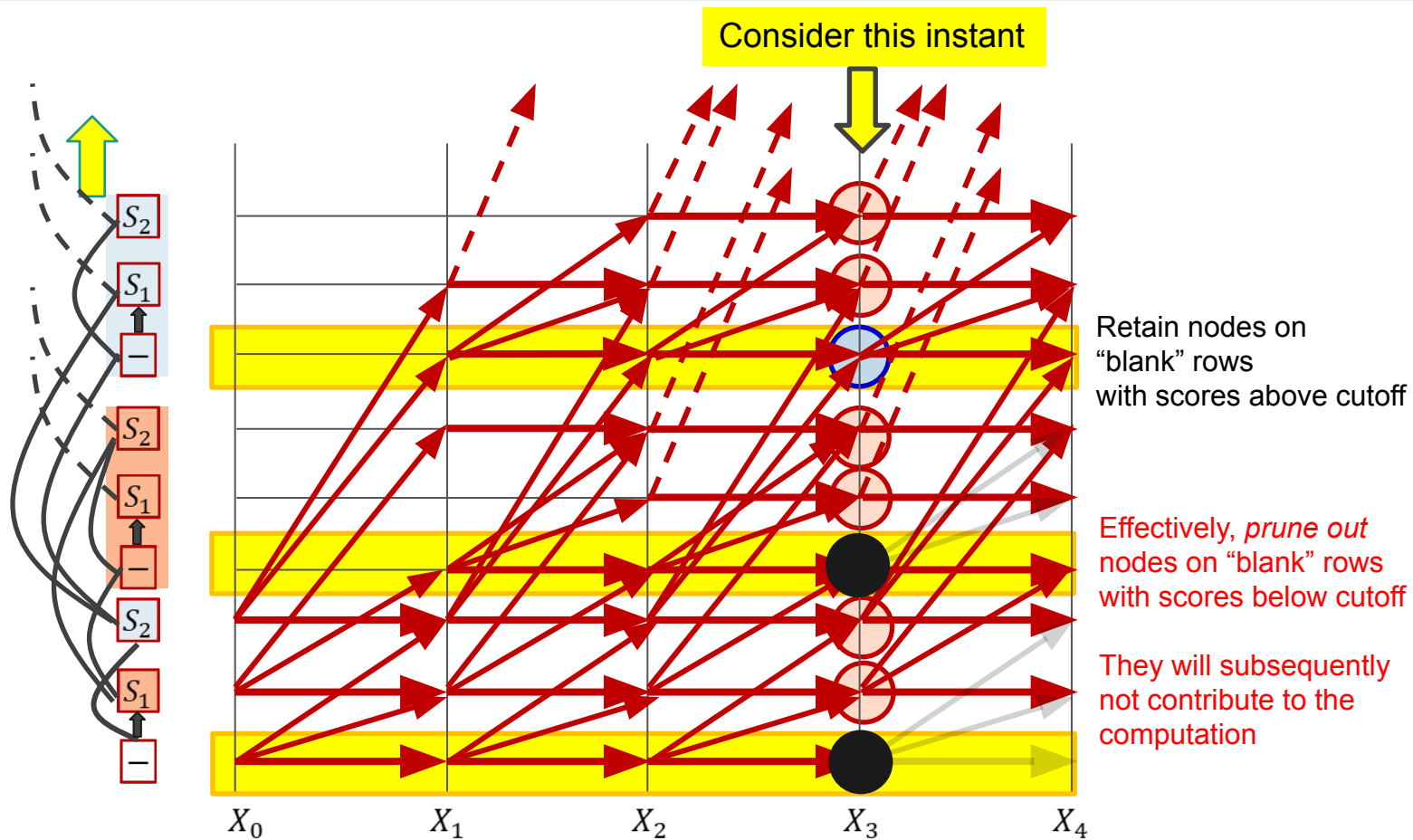
```
            PrunedPathScore[p] = PathScore[p]
```

```
        end
```

```
    end
```

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

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

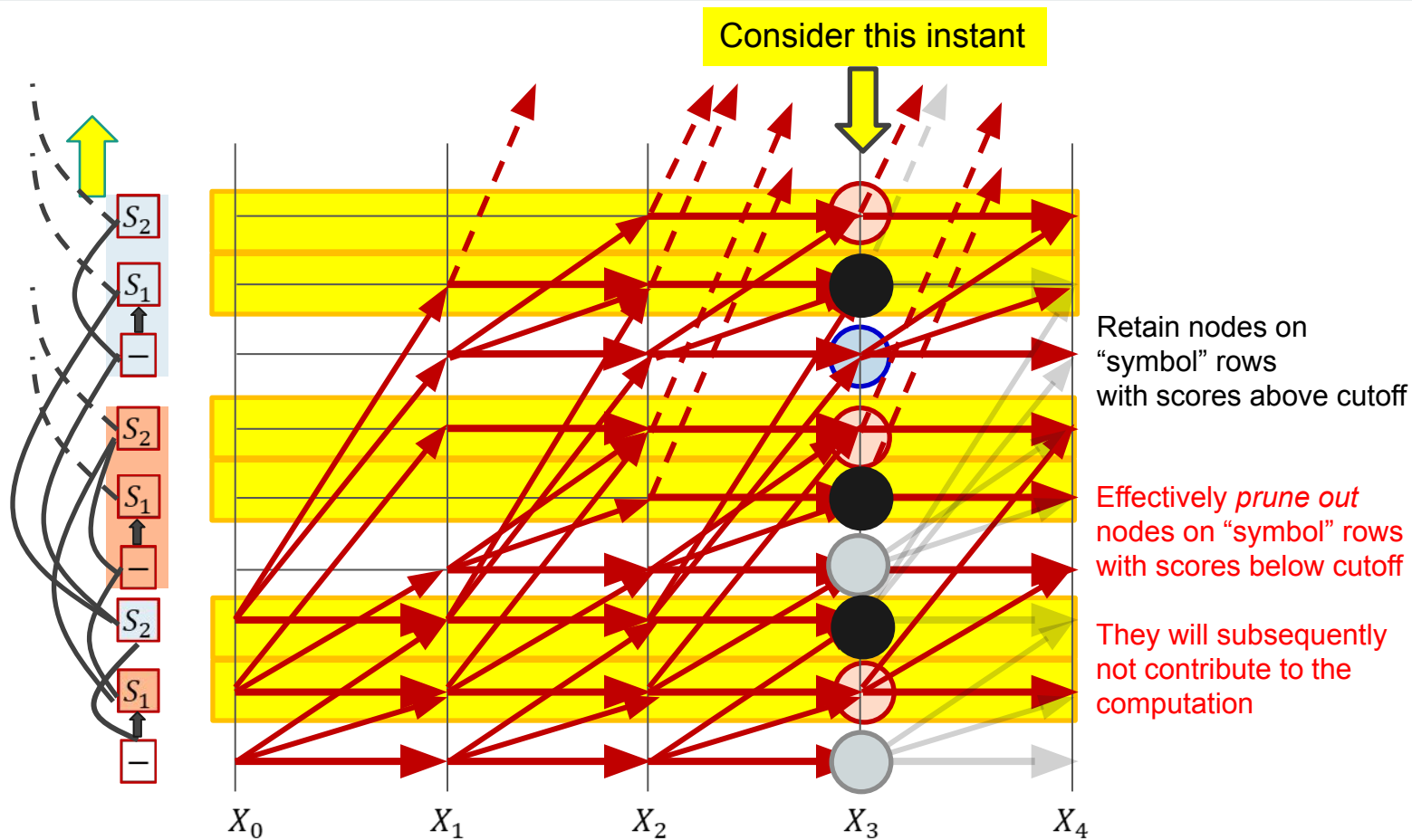
    # Sort and find cutoff score that retains exactly BeamWidth paths
    sort(scorelist) # In decreasing order
    cutoff = BeamWidth < length(scorelist) ? scorelist[BeamWidth] : scorelist[end]

    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
```

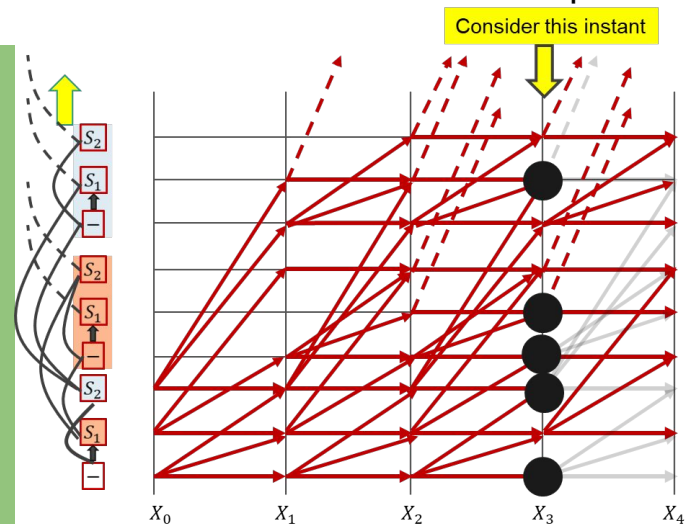
```
            PrunedPathScore[p] = PathScore[p]
```

```
        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,
        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)

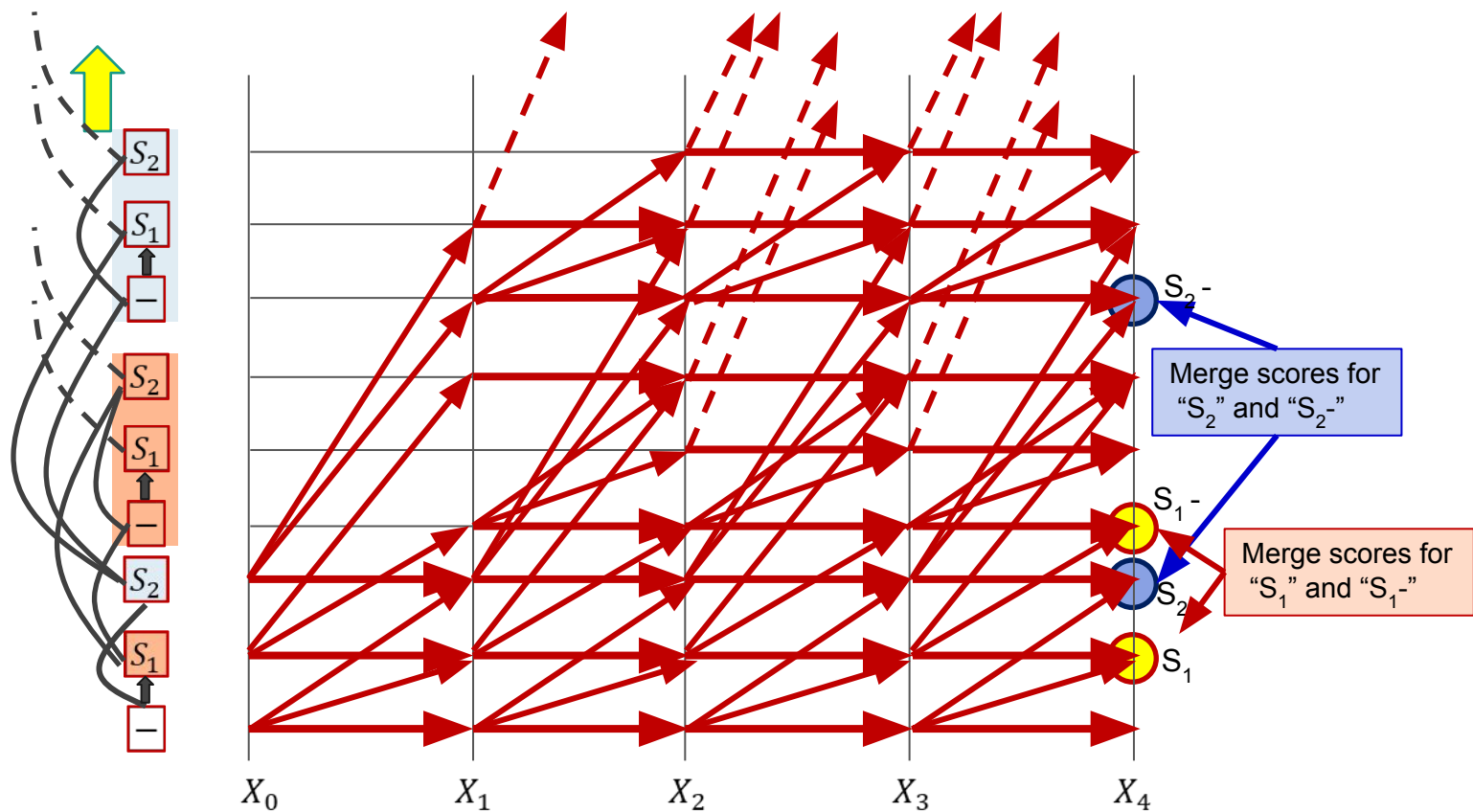
    # First extend paths by a blank
    NewPathsWithTerminalBlank, NewBlankPathScore = ExtendWithBlank(PathsWithTerminalBlank,
                                                                    PathsWithTerminalSymbol, y[:,t])

    # 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
```