A Probabilistic Head-corner Chart Parser for Tree Adjoining Grammars

Anoop Sarkar

March 29, 2000

```
Data Structures:

State = <n, goal, pos, fl, fr, type, {b, c, Pr}>
n:      node
goal:   goal node
pos:    {top, bot}
fl, fr: foot span
type:   {init, goal, item}
b:      backpointer to State
c:      backpointer to State
Pr:     probability of State when backpointers are b,c
{b,c,Pr}:  backpointer list (blist); corresponds to the n-best State list

Proc  = <i, j, State>
Slist = { s | s is State }
Plist = { p | p is Proc }
Chart = A{len, len} with a_{i,j} = Slist

All Elementary trees are assumed to be binary branching.

Functions:

add ({i,j}, State)
  - adds State to {i,j} only if it does not exist. Also State.type =
    init is taken to be equal to goal for the equality test. If the
    State exists but the backpointers are not in the blist append
    the new backpointers to the blist. If prob(.) is being used then
    use cmbprob(State,Pr) to update the state probability and sort
    blist to keep the most probable backpointers at head of list.
    Previous prob values can be stored with the blist to compute
    n-best values.
```

```
(State) exists in ({i,j})
  - operator that takes a State and checks chart at {i,j} and returns
    Slist or State of matches

anchors_for_tree(tree, index)
  - takes a tree and an word index and returns anchor nodes of the
    tree if tree is lexicalized by word at index

init_rootnodes(label)
  - takes a label and returns all initial trees rooted by nodes with
    that label

aux_rootnodes(label)
  - takes a label and returns all auxiliary trees rooted by nodes with
    that label

is_adjoinable(node)
  - returns true if an adjunction is possible at this node: rules out
    nodes like subst nodes, terminal nodes, footnodes, depending on
    the defn of adjunction used

is_subst(node)
  - returns true if node is a subst node

headtype(State)
  - if State.pos eq bot return ADJOIN
    else return context around node
    possible contexts = { UNARY_HEAD, LEFT_HEAD, RIGHT_HEAD }

nodetype(node)
  - return type of node s.n
    possible types = { ANCHOR, TERMINAL, SUBST, EPS, FOOT }

headcorner(node)
  - returns distinguished node called headcorner. The headcorner for
  the rootnode of auxiliary trees is always the footnode; for initial
  trees it is the (leftmost) anchor. For nodes other than the rootnode
  the headcorner is picked recursively from the ranked list in the defn
  of nodetype(node).

comptype(node)
  - completer type, one of the following:
      COMPL_INIT: root of initial tree
      COMPL_AUX: root of auxiliary tree
      COMPL_INTERNAL: otherwise; goal was internal to a tree
```

```
prob(node, tree, node.lex, tree.lex)
  - returns probability of adj/subst used for beam search

cmbprob(State, pr)
  - if (pr > State->pr) then State->pr = pr
```

```
Algorithm HParse:

__BEGIN__

N <- init_rootnodes('S')
start <- 0
len <- length(sent)+1
foreach n in N
     Plist <- add({start,len}, <n, n, top, _, _, init, _, _>)

foreach p in Plist
  {
     N1list <- init_head(p)
     N2list <- move_up(p)
     N3list <- completer(p)
     Plist <- { N1list , N2list , N3list }
  }

get_derivations('S')

__END__
```

```
init_head (p)
{
  s <- p.State
  if (type(s.n) neq init) { return }

  switch(nodetype(hc <- headcorner(s.n))) {

  case ANCHOR/TERMINAL:
     for (k = p.i; k < p.j; k++)
       N <- anchors_for_tree(tree(hc), k)
       foreach n in N
          Plist <- add({k,k+1}, <n, s.goal, bot, _, _, item, _, _>)

  case EPS:
     if (p.i eq p.j)
        Plist <- add({p.i,p.j}, <s.n, s.goal, top, _, _, item, _, _>)

  case SUBST:
     N <- init_rootnodes(label(hc))
     foreach n in N
        if (b <- <n, n, top, _, _, item, _, _> exists in {p.i, p.j})
           Pr <- prob(hc, tree(b.n), s.lex, b.lex)
           Plist <- add({p.i, p.j}, <hc, s.goal, top, _, _, item, b, 0>)
        else
           Plist <- add({p.i, p.j}, <hc, s.goal, top, _, _, goal, _, _>)
           Plist <- add({p.i, p.j}, <n, n, top, _, _, init, _, _>)

  case FOOT:
     Plist <- add({s.fl, s.fr}, <hc, s.goal, bot, s.fl, s.fr, item, _, _>)

  }
  s.type <- goal
  return(Plist)
}
```

```
move_up (p)
{
  s <- p.State
  g <- s.goal
  if (s.type neq item) OR ((s.pos eq top) AND (s.n eq g)) { return }
  pt <- parent(s.n)
  switch(headtype(s)) {

  case ADJOIN:
     Plist <- add({p.i,p.j}, <s.n, g, top, s.fl, s.fr, item, s, _>)

     if is_adjoinable(s.n)
        N <- aux_rootnodes(label(s.n))
        for (i = start; i <= p.i; i++)
            for (j = p.j; j < len; j++)
                if (i eq p.i) AND (j eq p.j) { continue }
                foreach n in N
                   if (b <- <n, n, top, p.i, p.j, item, _, _> exists in {i,j})
                      Pr <- prob(s.n, tree(b.n), s.lex, b.lex)
                      Plist <- add({i,j}, <s.n, g, top, p.i, p.j, item, b, s>)
                   else
                      Plist <- add({i,j}, <n, n, top, p.i, p.j, init, _, _>)

  case UNARY_HEAD:
     Plist <- add({p.i, p.j}, <pt, g, bot, s.fl, s.fr, item, s, _>)

  case LEFT_HEAD:
     r <- rightnode(s.n)
     for (k = p.j; k < len; k++)
         if (b <- <r, r, top, _, _, item, _> exists in {p.j, k})
            Plist <- add({p.i, k}, <pt, g, bot, s.fl, s.fr, item, s, b>)
         else
            Plist <- add({p.j, k}, <r, r, top, _, _, init, _, _>)

  case RIGHT_HEAD:
     l <- leftnode(s.n)
     for (k = start; k <= p.i; k++)
         if (b <- <l, l, top, _, _, item, _> exists in {k, p.i})
            Plist <- add({k, p.j}, <pt, g, bot, s.fl, s.fr, item, b, s>)
         else
            Plist <- add({k, p.i}, <l, l, top, _, _, init, _, _>)
  }
  return(Plist)
}
```

```
completer (p)
{
  s <- p.State
  if (s.type neq item) OR (s.pos neq top) OR (s.n neq s.goal) { return }
  switch(comptype(s.n)) {

  case COMPL_INIT:
    Slist <- { <n, ?goal, top, _, _, goal, _> |
                   label(s.n) eq label(n),
                   is_subst(n) } exists in {p.i, p.j}
    for b in Slist
        Pr <- prob(s.n, tree(b.n), s.lex, b.lex)
        Plist <- add({p.i, p.j}, <n, b.goal, top, _, _, item, s, 0>)

  case COMPL_AUX:
    Slist <- { <n, ?n.goal, bot, ?fl, ?fr, item, _> |
                   <footnode(tree(s.n)), _, bot, fl, fr, item, _> in {fl,fr},
                   label(s.n) eq label(n),
                   is_adjoinable(n) } exists in {s.fl, s.fr}
    for b in Slist
        Pr <- prob(s.n, tree(b.n), s.lex, b.lex)
        Plist <- add({p.i, p.j}, <n, b.goal, top, b.fl, b.fr, item, s, b>)

  case COMPL_INTERNAL:
    pt <- parent(s.n)
    switch(headtype(s.n)) {

    case LEFT_HEAD:
      r <- rightnode(s.n)
      for (k = p.j; k < len; k++)
          Slist <- { <r, ?goal, top, ?fl, ?fr, item, _> } exists in {p.j, k}
          for b in Slist
              Plist <- add({p.i, k}, <pt, b.goal, bot, b.fl, b.fr, item, s, b>)

    case RIGHT_HEAD:
      l <- leftnode(s.n)
      for (k = start; k <= p.i; k++)
          Slist <- { <l, ?goal, top, ?fl, ?fr, item, _> } exists in {k, p.i}
          for b in Slist
              Plist <- add({k, p.j}, <pt, b.goal, bot, b.fl, b.fr, item, b, s>)
    }
  }
  return(Plist)
}
```