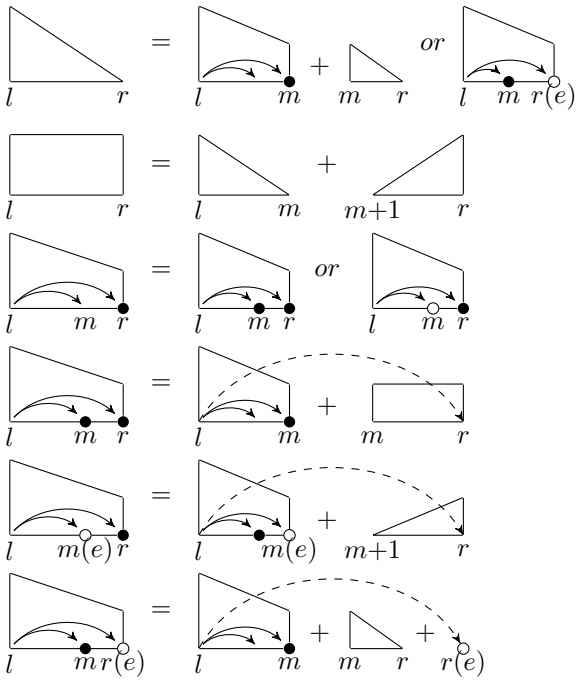
Supplementary Note

February 7, 2017



Completed Constituent(CC)

Juxtaposed Constituent(JC)

Solid Outside Constituent(SOC)

Solid Both Constituent(SBC)

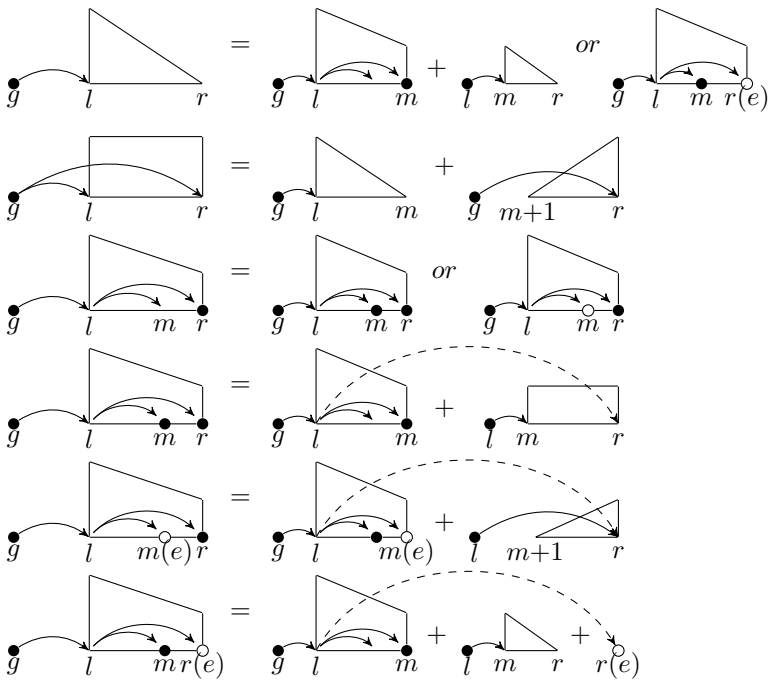
Empty Inside Constituent(EIC)

Empty Outside Constituent(EOC)

Algorithm 1 Empty Category Second Order Eisner Algorithm

procedure EC2NDDECODE($sent$) ▷ decode the sentence
  $len \leftarrow$ **len**($sent$)
  for $d \leftarrow 1, len+1$ **do** ▷ distance
    for $l \leftarrow 0, len-d+1$ **do** ▷ span's left
      $r \leftarrow l+d-1$ ▷ span's right
      $JC[l][r] \leftarrow \max_{m \in (l,r)} CC[l][m][0]+CC[m+1][r][1]$

      $EIC[l][r][0] \leftarrow \max_{m \in (l,r)} EOC[l][m][0]+CC[m+1][r][1]$
      $SBC[l][r][0] \leftarrow \max_{m \in (l,r)} SOC[l][m][0]+JC[m][r]$
      $EOC[l][r][0] \leftarrow \max_{m \in (l,r)} SOC[l][m][0]+CC[m][r][0]+arc(l,r(e))$
      $SOC[l][r][0] \leftarrow \max(SBC[l][r][0], EIC[l][r][0])$
      $CC[l][r][0] \leftarrow \max_{m \in (l,r)} SOC[l][m][0]+CC[m][r][0]$
      $CC[l][r][0] \leftarrow \max(CC[l][r][0], SOC[l][r][0], EOC[l][r][0])$
      ▷ left to right constituent
      $EIC[l][r][1] \leftarrow \max_{m \in (l,r)} EOC[l][m][1]+CC[m+1][r][0]$
      $SBC[l][r][1] \leftarrow \max_{m \in (l,r)} SOC[m][r][1]+JC[l][m]$
      $EOC[l][r][1] \leftarrow \max_{m \in (l,r)} SOC[m][r][0]+CC[l][m][1]+arc(l(e),r)$
      $SOC[l][r][1] \leftarrow \max(SBC[l][r][1], EIC[l][r][1])$
      $CC[l][r][1] \leftarrow \max_{m \in (l,r)} SOC[m][r][1]+CC[l][m][1]$
      $CC[l][r][1] \leftarrow \max(CC[l][r][1], SOC[l][r][1], EOC[l][r][1])$
      ▷ right to left constituent
    end for
  end for
  return
end procedure



For any arc, or, set of arcs in a dependency tree, we use their terminal token's word and POS tag as features.

We following eisner algorithm to select our features, and we divide our feature into four parts: f-arc, f-sibling, f-grand and f-grand-sibling

f-arc contains following features:

We use symbols w and p denote word and postag, and use h and c indicate head and child token.

h.w, h.p, c.w, c.p, h.wp, c.wp, h.wp+c.w, h.wp+c.p, c.wp+h.w, c.wp+h.p, h.wp+c.wp

h.p+(h+1).p+(c-1).p+c.p, (h-1).p+h.p+(c-1).p+c.p, h.p+(h+1).p+c.p+(c+1).p, (h-1).p+h.p+c.p+(c+1).p

h.p+c.p+b.p, b means token between h and c

Since we will add empty node during our decoding, and those new token will increase edges. We can see that if we add a new empty category, those unigram feature(such as h.w, c.p) will contribute to the total score without any side-effect. So the more empty node we have, the more unigram feature score we get. It's unreasonable. So we abandon unigram features when we add an empty token.

f-sibling contains following features:

Except for head and child tokens, we have middle(m) token here, which means the inner split point in Incompleted Constituent.

The features consist of middle and child tokens, and all three tokens' POS tags.

m.p+c.p, m.w+c.w, m.w+c.p, m.p+c.w, h.p+m.p+c.p

f-grand contains following features:

We use g stands for grand

The features consist of grand and child tokens, and all three tokens' POS tags

g.p+c.p, g.w+c.w, g.w+c.p, g.p+c.w, g.p+h.p+c.p

f-grand-sibling features:

The features consist of grand, child and middle tokens, and all four tokens.

g.w+h.p+m.p+c.p, g.p+h.w+m.p+c.p, g.p+h.p+m.w+c.p, g.p+h.p+m.p+c.w

g.w+h.w+m.p+c.p, g.w+h.p+m.w+c.p, g.w+h.p+m.p+c.w, g.p+h.w+m.w+c.p, g.p+h.w+m.p+c.w, g.p+h.p+m.w+c.w

g.p+h.p+m.p+c.p

g.p+m.p+c.p, g.p+m.p+c.w, g.p+m.w+c.p, g.w+m.p+c.p, g.w+m.w+c.p, g.w+m.p+c.w, g.p+m.w+c.w

For higher order sibling features, it will have more inner split points, those split points contribute more features in richer combinations of tokens.