# spellout.grm: finite-state spell-out transducers for English

##alphabets

symbol = \% | \_ | - | / | \ \ | ' | "3;";
V = \"A\" | \"E\" | \"I\" | \"O\" | \"U\";
C = \"B\" | \"C\" | \"D\" | \"F\" | \"G\" | \"H\" | \"J\" | \"K\" | \"L\" | \"M\" |
    \"N\" | \"P\" | \"Q\" | \"R\" | \"S\" | \"T\" | \"V\" | \"W\" | \"X\" | \"Y\" | \"Z\";
letter = V | C;
stemsym = Optimize[letter | symbol];

## suffixes

s  = \"/S\" | \"/3S\"; # since these always behave exactly the same
z  = \"/Z\";
ed  = \"/ED\";
ing = \"/ING\";
_d  = \"/\-'D\";
_m  = \"/\-'M\";
_s  = \"/\-'S\";
_t  = \"/\-'T\";
_ll = \"/\-'LL\";
_re = \"/\-'RE\";
_ve = \"/\-'VE\";
n_t = \"/N'T\";
suffix = Optimize[s | z | ed | ing | _d | _m | _s | _t | _ll | _re | _ve | n_t];
alphabet = Optimize[(stemsym | suffix)*];

## stem change rules

# y -> i, as in bury -> buried
y2i_suffix = s | ed;
y2i = Optimize[CDRewrite[\"Y\": \"IE\", C, y2i_suffix, alphabet]]; # ie ie2y = Optimize[CDRewrite[\"IE\": \"Y\", C, ing, alphabet]]; # e -> 0 / __ ing
e_exc = \"[BOS]\" (\"BE\" | \"EYE\")
e_before_ing = Optimize[CDRewrite[\"\": \"\", e_exc, ing, alphabet] @
    CDRewrite[\"E\": \"\", C, ing, alphabet] @
    CDRewrite[\"\": \"\", e_exc, ing, alphabet]];# f -> ve, as in wolf -> wolves
f2ve_words = \"CAL\" | \"EL\" | \"DWAR\" | \"HAL\" | \"HOO\" | \"LEA\" | \"LOA\" |
    \"SCAR\" | \"SEL\" | \"SHEL\" | \"STAF\" | \"THIE\" | \"WOL\";
f2ve = Optimize[CDRewrite[\"F\"{1,2}: \"VE\", f2ve_words, s, alphabet]];# doubling rules
doubling_left  = (C | \"QU\") V;
doubling_suffix = ing | ed;
func CopyRule[X, left_context, right_context, alphabet] {
    return CDRewrite[X: X X, left_context, right_context, alphabet];
}
double = Optimize[CopyRule[\"B\", doubling_left, doubling_suffix, alphabet] @
    CDRewrite[\"C\": \"CK\", V, doubling_suffix, alphabet] @
    CopyRule[\"D\", doubling_left, doubling_suffix, alphabet] @
    CopyRule[\"F\", doubling_left, doubling_suffix, alphabet] @
CopyRule["G", doubling_left, doubling_suffix, alphabet] @
CopyRule["M", doubling_left, doubling_suffix, alphabet] @
CopyRule["N", doubling_left, doubling_suffix, alphabet] @
CopyRule["P", doubling_left, doubling_suffix, alphabet] @
CopyRule["S", doubling_left, doubling_suffix, alphabet] @
CopyRule["T", doubling_left, doubling_suffix, alphabet] @
CopyRule["M", doubling_left, doubling_suffix, alphabet] @
CopyRule["N", doubling_left, doubling_suffix, alphabet] @
CopyRule["P", doubling_left, doubling_suffix, alphabet] @
CopyRule["S", doubling_left, doubling_suffix, alphabet] @
CopyRule["T", doubling_left, doubling_suffix, alphabet] @
# now, exceptions
CDRewrite["TT": "T", "VISI", doubling_suffix, alphabet] @
CDRewrite["NN": "N", "E", doubling_suffix, alphabet]
);

stem_rules = y2i @ ie2y @ f2ve @ double @ e_before_ing;
## suffix rules
es_cntx = ("CH" | "SH" | "S" | "Z" | "X" | "GO");
s_spellout = Optimize[CDRewrite["": "S", ",", s, alphabet] @
    CDRewrite["": "E", es_cntx, "S" s, alphabet]@
    ];
sz_coalescence = Optimize[CDRewrite["": ",", s z, alphabet]@
    ];
z_spellout = Optimize[CDRewrite["": ",", z, alphabet] @
    # this overgenerates when /z is preceded by an /s,
    # so we just undo that
    CDRewrite["": ",", s z, alphabet]@
    ];
ed_cntx = C | ("I" | "O");
ed_spellout = Optimize[CDRewrite["": ",", ed, alphabet] @
    CDRewrite["": ",", ed_cntx, "D" ed, alphabet]@
    ];
func SpelloutRule[string, suffix, alphabet] {
    return CDRewrite["": string, ",", suffix "][EOS]\", alphabet];
}
other_spellout = Optimize[SpelloutRule["ING", ing, alphabet] @
    SpelloutRule["": ",", _d, alphabet] @
    SpelloutRule["": ",", _m, alphabet] @
    SpelloutRule["": ",", _s, alphabet] @
    SpelloutRule["": ",", _t, alphabet] @
    SpelloutRule["": ",", _l, alphabet] @
    SpelloutRule["": ",", _re, alphabet] @
    SpelloutRule["": ",", _ve, alphabet] @
    SpelloutRule["": ",", n_t, alphabet]@
    ];
suffix_rules = s_spellout @ sz_coalescence @ z_spellout @ ed_spellout @
other_spellout;

## putting it all together
export spellout = Optimize[stem_rules @ suffix_rules];