ENV.defaults["stage.limitIters"] = 50

tokens = { type: "CLASS", value: "edu.stanford.nlp.ling.CoreAnnotations$TokensAnnotation" }

numtokens = { type: "CLASS", value: "edu.stanford.nlp.ling.CoreAnnotations$NumerizedTokensAnnotation" }

$TIMEOFDAY = "/morning|afternoon|evening|night|noon|midnight|teatime|lunchtime|dinnertime|suppertime|afternoon|midday|dusk|dawn|sunup|sunrise|sundown|twilight|daybreak/";

$NUM = ( [ { numcomptype:NUMBER } ] );

$INT = ( [ { numcomptype:NUMBER } & !{ word:/.\*\.\d+.\*/} & !{ word:/.\*,.\*/ } ] );

$INTORD = ( [ ({ numcomptype:NUMBER } | { numcomptype:ORDINAL }) & !{ word:/.\*\.\d+.\*/} & !{ word:/.\*,.\*/ } ] );

$INT1000TO3000 = ( [ $INT & !{ word:/\+.\*/} & { numcompvalue>1000 } & { numcompvalue<3000 } ] );

$INT1TO31 = ( [ $INTORD & !{ word:/\+.\*/} & { numcompvalue>=1 } & { numcompvalue<=31 } ] );

$NUM\_ORD = ( [ { numcomptype:ORDINAL } ] );

$INT\_TIMES = ( $INT /times/ | once | twice | thrice );

$REL\_MOD = ( /the/? /next|following|last|previous/ | /this/ /coming|past/? | /the/ /coming|past/ );

$FREQ\_MOD = ( /each/ | /every/ $NUM\_ORD | /every/ /other|alternate|alternating/? | /alternate|alternating/ );

$EARLY\_LATE\_MOD = ( /late|early|mid-?/ | /the/? /beginning|start|dawn|middle|end/ /of/ );

$APPROX\_MOD = ( /about|around|some|exactly|precisely/ );

$YEAR = ( /[012]\d\d\d/ | /'\d\d/ | /'/ /\d\d/ | /\w+teen|twenty/ [ { numcompvalue<100 } & { numcompvalue>0 } & $INT ] );

$POSSIBLE\_YEAR = ( $YEAR /a\.?d\.?|b\.?c\.?/? | $INT /a\.?d\.?|b\.?c\.?/ | $INT1000TO3000 );

$hasTemporal = "( { temporal::EXISTS } & {{ temporal.value != NON\_TEMPORAL }} & !{{ tags[\"TIMEZONE\"] }} )"

# Decades

DECADES\_MAP = {

"twenties": "2X",

"thirties": "3X",

"forties": "4X",

"fifties": "5X",

"sixties": "6X",

"seventies": "7X",

"eighties": "8X",

"nineties": "9X"

}

$Decades = CreateRegex(Keys(DECADES\_MAP))

# Durations

TIMEUNIT\_MAP = {

"year": YEAR,

"yr": YEAR,

"month": MONTH,

"mo": MONTH,

"day": DAY,

"hour": HOUR,

"hr": HOUR,

"minute": MINUTE,

"min": MINUTE,

"second": SECOND,

"sec": SECOND,

"millisecond": MILLIS,

"millisec": MILLIS,

"week": WEEK,

"wk": WEEK,

"fortnight": FORTNIGHT,

"quarter": QUARTER,

"century": CENTURY,

"centuries": CENTURY,

"millennia": MILLENNIUM,

"millennium": MILLENNIUM,

"millenia": MILLENNIUM,

"millenium": MILLENNIUM

}

$TEUnits = CreateRegex(Keys(TIMEUNIT\_MAP))

BASIC\_NUMBER\_MAP = {

"one": 1,

"two": 2,

"three": 3,

"four": 4,

"five": 5,

"six": 6,

"seven": 7,

"eight": 8,

"nine": 9,

"ten": 10,

"eleven": 11,

"twelve": 12,

"thirteen": 13,

"fourteen": 14,

"fifteen": 15,

"sixteen": 16,

"seventeen": 17,

"eighteen": 18,

"nineteen": 19,

"twenty": 20,

"thirty": 30,

"forty": 40,

"fifty": 50,

"sixty": 60,

"seventy": 70,

"eighty": 80,

"ninety": 90,

"hundred": 100

}

$BasicNumTerm = CreateRegex(Keys(BASIC\_NUMBER\_MAP))

BASIC\_ORDINAL\_MAP = {

"first": 1,

"second": 2,

"third": 3,

"fourth": 4,

"fifth": 5,

"sixth": 6,

"seventh": 7,

"eighth": 8,

"ninth": 9,

"tenth": 10,

"eleventh": 11,

"twelfth": 12,

"thirteenth": 13,

"fourteenth": 14,

"fifteenth": 15,

"sixteenth": 16,

"seventeenth": 17,

"eighteenth": 18,

"nineteenth": 19,

"twentieth": 20,

"thirtieth": 30,

"fortieth": 40,

"fiftieth": 50,

"sixtieth": 60,

"seventieth": 70,

"eightieth": 80,

"ninetieth": 90,

"hundredth": 100

}

$BasicOrdTerm = CreateRegex(Keys(BASIC\_ORDINAL\_MAP))

########################################################################################################################

ENV.defaults["stage"] = 0

ENV.defaults["ruleType"] = "tokens"

{ pattern: ( $POSSIBLE\_YEAR ),

action: (

Tag($0, "YEAR",

:case {

$0 =~ ( /\w+teen|twenty/ [ $INT ] ) => Add(Multiply($0[0].numcompvalue, 100), $0[1].numcompvalue),

$0 =~ ( /'/ /\d\d/ ) => Concat("XX", $0[1].word),

$0 =~ ( /'\d\d/ ) => Concat("XX", $0[0].word.substring(1)),

:else => $0[0].numcompvalue

}

),

Tag($0, "YEAR\_ERA",

:case {

$0 =~ ( $INT /a\.?d\.?/ ) => ERA\_AD,

$0 =~ ( $INT /b\.?c\.?/ ) => ERA\_BC,

:else => ERA\_UNKNOWN

}

)

)

}

# Operators

{ pattern: ( /this/ ),

action: Tag($0, "TemporalOp", THIS) }

{ pattern: ( /next/ ),

action: Tag($0, "TemporalOp", NEXT) }

{ pattern: ( /following/ ),

action: Tag($0, "TemporalOp", NEXT) }

{ pattern: ( /previous/ ),

action: Tag($0, "TemporalOp", PREV) }

{ pattern: ( /last/ ),

action: Tag($0, "TemporalOp", PREV) }

{ pattern: ( /this|the/ /coming|following|next/ ),

action: Tag($0, "TemporalOp", NEXT\_IMMEDIATE) }

{ pattern: ( /this|the/ /past|previous|last/ ),

action: Tag($0, "TemporalOp", PREV\_IMMEDIATE) }

# Modifiers

# Early late modifiers

{ pattern: ( /late/ | /end/ ),

action: Tag($0, "Modifier", "LATE") }

{ pattern: ( /early/ | /beginning|start|dawn/ ),

action: Tag($0, "Modifier", "EARLY") }

{ pattern: ( /mid-?/ | /middle/ ),

action: Tag($0, "Modifier", "MID") }

# Frequency modifiers

{ pattern: ( /each/ | /every/ ),

action: ( Tag($0, "PTS.quant", $0), Tag($0, "PTS.multiple", 1 ) ) }

{ pattern: ( /every/ ($NUM\_ORD|$INT) ),

action: ( Tag($0, "PTS.quant", $0), Tag($0, "PTS.multiple", $1[0].numcompvalue ) ) }

{ pattern: ( /every/ /other|alternate|alternating/ | /alternate|alternating/ ),

action: ( Tag($0, "PTS.quant", $0), Tag($0, "PTS.multiple", 2 ) ) }

# Approximate modifiers

{ pattern: ( /about|around|some/ ),

action: Tag($0, "Modifier", "APPROX") }

{ pattern: ( /exactly|precisely/ ),

action: Tag($0, "Modifier", "EXACT") }

# Periodic Set

PERIODIC\_SET = {

"centennial": TemporalCompose(MULTIPLY, YEARLY, 100),

"yearly": YEARLY,

"annually": YEARLY,

"annual": YEARLY,

"hourly": HOURLY,

"nightly": NIGHTLY,

"daily": DAILY,

"weekly": WEEKLY,

"monthly": MONTHLY,

"quarterly": QUARTERLY

}

$PeriodicSetRegex = CreateRegex(Keys(PERIODIC\_SET))

{ pattern: ( /.\*($PeriodicSetRegex)/ ),

matchWithResults: TRUE,

action: Tag($0, "PeriodicSet", PERIODIC\_SET[Lowercase($$0.matchResults[0].word.group(1))]) }

########################################################################################################################

ENV.defaults["stage"] = 1

ENV.defaults["ruleType"] = "text"

# Durations: 3-months old or three-months old

{ text: /(\d+)[-]($TEUnits)(s)?([-\s]old)?/ => Duration( TIMEUNIT\_MAP[Lowercase($2)], $1 ) }

{ text: /($BasicNumTerm)[-]($TEUnits)(s)?([-\s]old)?/ =>

Duration( TIMEUNIT\_MAP[Lowercase($2)], BASIC\_NUMBER\_MAP[Lowercase($1)] ) }

# Durations

ENV.defaults["ruleType"] = "tokens"

{ (/years?|yrs?/) => YEAR }

{ (/months?|mos?/) => MONTH }

{ (/days?/) => DAY }

{ (/hours?|hrs?/) => HOUR }

{ (/minutes?|mins?/) => MINUTE }

{ (/seconds?|secs?/) => SECOND }

{ (/milliseconds?|millisecs?/) => MILLIS }

{ (/weeks?|wks?/) => WEEK }

{ (/fortnights?/) => FORTNIGHT }

{ (/quarters?/) => QUARTER }

{ (/decades?/) => DECADE }

{ (/centurys?|centuries?/) => CENTURY }

{ (/millenn?ias?|millenn?iums?/) => MILLENNIUM }

# Time of Day

{ (/mornings?/) => MORNING }

{ (/afternoons?/) => AFTERNOON }

{ (/evenings?/) => EVENING }

{ (/dusks?/) => DUSK }

{ (/twilights?/) => TWILIGHT }

{ (/dawns?|daybreaks?/) => DAWN }

{ (/sunrises?|sunups?/) => SUNRISE }

{ (/sundowns?|sunsets?/) => SUNSET }

{ (/middays?|noons?/) => NOON }

{ (/midnights?/) => MIDNIGHT }

{ (/teatimes?/) => TEATIME }

{ (/lunchtimes?/) => LUNCHTIME }

{ (/dinnertimes?/) => DINNERTIME }

{ (/suppertimes?/) => SUPPERTIME }

{ (/daylights?|days?|daytimes?/) => DAYTIME }

{ (/nighttimes?|nights?|overnights?/) => NIGHT }

# Seasons

{ (/summers?/) => SUMMER }

{ (/winters?/) => WINTER }

{ (/falls?|autumns?/) => FALL }

{ (/springs?/) => SPRING }

# Relative times

{ (/yesterdays?/) => YESTERDAY }

{ (/todays?/) => TODAY }

{ (/tomorrows?/) => TOMORROW }

{ (/tonights?|tonites?/) => TONIGHT }

# Day of Week

{ (/mondays?/) => MONDAY }

{ (/tuesdays?/) => TUESDAY }

{ (/wednesdays?/) => WEDNESDAY }

{ (/thursdays?/) => THURSDAY }

{ (/fridays?/) => FRIDAY }

{ (/saturdays?/) => SATURDAY }

{ (/sundays?/) => SUNDAY }

{ (/mons?/) => MONDAY }

{ (/tues?/) => TUESDAY }

{ (/weds?/) => WEDNESDAY }

{ (/thurs?/) => THURSDAY }

{ (/fris?/) => FRIDAY }

{ (/sats?/) => SATURDAY }

{ (/suns?/) => SUNDAY }

{ (/weekends?/) => WEEKEND }

{ (/weekdays?/) => WEEKDAY }

# Month

{ (/januarys?/) => JANUARY }

{ (/februarys?/) => FEBRUARY }

{ (/marchs?/) => MARCH }

{ (/aprils?/) => APRIL }

{ (/mays?/) => MAY }

{ (/junes?/) => JUNE }

{ (/julys?/) => JULY }

{ (/augusts?/) => AUGUST }

{ (/septembers?/) => SEPTEMBER }

{ (/octobers?/) => OCTOBER }

{ (/novembers?/) => NOVEMBER }

{ (/decembers?/) => DECEMBER }

{ (/jan\.?/) => JANUARY }

{ (/feb\.?/) => FEBRUARY }

{ (/mar\.?/) => MARCH }

{ (/apr\.?/) => APRIL }

{ (/jun\.?/) => JUNE }

{ (/jul\.?/) => JULY }

{ (/aug\.?/) => AUGUST }

{ (/sept?\.?/) => SEPTEMBER }

{ (/oct\.?/) => OCTOBER }

{ (/nov\.?/) => NOVEMBER }

{ (/dec\.?/) => DECEMBER }

{ ruleType: "filter",

over: NIL,

pattern: ( [ { temporal::IS\_TIMEX\_DATE } & {{ tokens[0].tag =~ /NN.\*S/ }} ] ),

result: MakePeriodicTemporalSet($0[0].temporal.value)

}

########################################################################################################################

ENV.defaults["ruleType"] = "tokens"

ENV.defaults["priority"] = 0

ENV.defaults["locale"] = "en"

// Military times with time zones from http://www.timeanddate.com/library/abbreviations/timezones/military/

MILITARY\_TIME\_ZONE\_MAP = {

"A": 1,

"B": 2,

"C": 3,

"D": 4,

"E": 5,

"F": 6,

"G": 7,

"H": 8,

"I": 9,

"K": 10,

"L": 11,

"M": 12,

"N": -1,

"O": -2,

"P": -3,

"Q": -4,

"R": -5,

"S": -6,

"T": -7,

"U": -8,

"V": -9,

"W": -10,

"X": -11,

"Y": -12,

"Z": 0

}

{

ruleType: "tokens",

pattern: ( (/(\d\d)(\d\d)([A-Z])/) ),

matchWithResults: TRUE,

result: { type: "Temporal",

value: IsoTime($$1.matchResults[0].word.group(1), $$1.matchResults[0].word.group(2), NIL).setTimeZone(

MILITARY\_TIME\_ZONE\_MAP[$$1.matchResults[0].word.group(3)] ) },

// Change to TRUE to support military time zones

active: FALSE

}

# ISO date/times

# TODO: Support other timezone formats

{ ruleType: "time", pattern: /yyyy-?MM-?dd-?'T'HH(:?mm(:?ss([.,]S{1,3})?)?)?(Z)?/ }

{ ruleType: "time", pattern: /yyyy-MM-dd/ }

{ ruleType: "time", pattern: /'T'HH(:?mm(:?ss([.,](S{1,3}))?)?)?(Z)?/ }

# Tokenizer "sometimes adds extra slash

{ ruleType: "time", pattern: /yyyy\\?\/MM\\?\/dd/ }

{ ruleType: "time", pattern: /MM?\\?\/dd?\\?\/(yyyy|yy)/ }

{ ruleType: "time", pattern: /MM?-dd?-(yyyy|yy)/ }

{ ruleType: "time", pattern: /HH?:mm(:ss)?(Z)?/ }

{ ruleType: "time", pattern: /yyyy-MM/ }

# Euro - Ambiguous pattern - interpret as dd.MM.yy(yy)

{ ruleType: "time", pattern: /dd?\.MM?\.(yyyy|yy)/ }

{ ruleType: "time", pattern: /HH?''hmm/ }

# Timezones

{ ruleType: "time", pattern: /zzz/, action: Tag(\_, "TIMEZONE", TRUE) }

{ ruleType: "time", pattern: /ZZZ/, action: Tag(\_, "TIMEZONE", TRUE) }

# Birthdays

{ ( [ { tag:NNP } ]+ [ { tag:POS } ] /birthday/ ) => SimpleTime($0) }

# Generic decade

{ ( /the/? ( /\w+teen/ /$Decades/ ) )

=> IsoDate( Concat( Format("%02d", $0[0].numcompvalue), DECADES\_MAP[Lowercase($0[1].word)]), NIL, NIL)

}

{ ( /the/? ( /$Decades/ ) )

=> IsoDate( Concat("XX", DECADES\_MAP[Lowercase($0[0].word)]) , NIL, NIL)

}

{ (/the/? /'/ /\d\d/ ) => IsoDate( Format( "XX%02d", $0[-1].numcompvalue), NIL, NIL) }

{ (/the/? /'/ /\d0s/ | /the/? /'\d0s/ ) => IsoDate( Replace($0[-1].word, /'?(\d)0s/, "XX$1X"), NIL, NIL) }

{ (/the/? /\d\d\d0s/) => IsoDate( Replace($0[-1].word, /(\d\d\d)0s/, "$1X"), NIL, NIL) }

{ (/the/? /\d\d00s/) => IsoDate( Replace($0[-1].word, /(\d\d)00s/, "$1XX"), NIL, NIL) }

{ (/the/? /mid-\d\d\d0s/) => IsoDate( Replace($0[-1].word, /mid-(\d\d\d)0s/, "$1X"), NIL, NIL) }

{ (/the/? /mid-\d\d00s/) => IsoDate( Replace($0[-1].word, /mid-(\d\d)00s/, "$1XX"), NIL, NIL) }

# some century expressions

{ ( (/every/ $NUM\_ORD) (/centurys?|ies/) ) =>

MakePeriodicTemporalSet(CENTURY, GetTag($1[0], "PTS.quant"), GetTag($1[0], "PTS.multiple") ) }

{ ( /the/? ($NUM\_ORD) /-/? /century/ (/b\.?c\.?/) )

=> IsoDate(

Format("-%02dXX", Subtract($1[0].numcompvalue, 1)),

NIL, NIL)

}

{ pattern: ( /the/? (/($BasicOrdTerm)-century/) (/b\.?c\.?/) ),

matchWithResults: TRUE,

result: IsoDate(

Format("-%02dXX", Subtract(BASIC\_ORDINAL\_MAP[Lowercase($$1.matchResults[0].word.group(1))], 1)),

NIL, NIL)

}

{ pattern: ( /the/? (/(\d+)(st|nd|rd|th)-century/) (/b\.?c\.?/) ),

matchWithResults: TRUE,

result: IsoDate(

Format("-%02dXX", Subtract( { type: "NUMBER", value: $$1.matchResults[0].word.group(1) }, 1)),

NIL, NIL)

}

{ ( /the/? ($NUM\_ORD) /-/? /century/ (/a\.?d\.?/)? )

=> IsoDate(

Format("%02dXX", Subtract($1[0].numcompvalue, 1)),

NIL, NIL)

}

{ pattern: ( /the/? (/($BasicOrdTerm)-century/) (/a\.?d\.?/)? ),

matchWithResults: TRUE,

result: IsoDate(

Format("%02dXX", Subtract(BASIC\_ORDINAL\_MAP[Lowercase($$1.matchResults[0].word.group(1))], 1)),

NIL, NIL)

}

{ pattern: ( /the/? (/(\d+)(st|nd|rd|th)-century/) (/a\.?d\.?/)? ),

matchWithResults: TRUE,

result: IsoDate(

Format("%02dXX", Subtract( { type: "NUMBER", value: $$1.matchResults[0].word.group(1) }, 1)),

NIL, NIL)

}

# some quarter expressions - need to add year refs

{ ( /the/? [{tag:JJ}]? ($NUM\_ORD) /-/? [{tag:JJ}]? /quarter/ ) =>

TemporalCompose(CREATE, QUARTER\_OF\_YEAR, $1[0].numcompvalue) }

{ text: /(\d+)(st|nd|rd|th)-quarter/ =>

TemporalCompose(CREATE, QUARTER\_OF\_YEAR, $1 )

}

{ text: /($BasicOrdTerm)-quarter/ =>

TemporalCompose(CREATE, QUARTER\_OF\_YEAR, BASIC\_ORDINAL\_MAP[Lowercase($1)])

}

# (unit)ly

{ ruleType: "tokens",

# pattern: ( (?m){1,3} /((bi|semi)\s\*-?\s\*)?((annual|year|month|week|dai|hour|night|quarter)ly|annual)/ ),

pattern: ( (?m){1,3} /((bi|semi)\s\*-?\s\*)?($PeriodicSetRegex)/ ),

result: :case {

$0[0].word =~ /bi.\*/ => TemporalCompose(MULTIPLY, GetTag($0[-1], "PeriodicSet"), 2),

$0[0].word =~ /semi.\*/ => TemporalCompose(DIVIDE, GetTag($0[-1], "PeriodicSet"), 2),

:else => GetTag($0[-1], "PeriodicSet") }

}

# some interval expressions

{ text: /\b(\d{4})\s\*(?:-)\s\*(\d{4})\b/ =>

TimeRange( IsoDate($1, NIL, NIL), IsoDate($2, NIL, NIL) ) }

{ ( /the/ /weekend/ ) => WEEKEND }

# Now a few time expressions

{ ( (/\d\d\d\d/) /hours?/? (/universal|zulu/ | /[a-z]+/ /standard|daylight/) /time/ ) => IsoTime($1[0].word, NIL, NIL) }

{ ( (/\d\d?/) /hours?/ (?: (/\d\d?/) /minutes?/?)? (/universal|zulu/ | /[a-z]+/ /standard|daylight/) /time/ )

=> IsoTime($1[0].word, $1[0].word, NIL) }

{ text: /(\d\d):?(\d\d)(:?(\d\d))?\s\*h(ou)rs?/ => IsoTime($1,$2,$3) }

{ text: /(\d\d?)(:?(\d\d))(:\d\d)?a\.?m\.?/ => TemporalCompose(INTERSECT, IsoTime($1,$3,$4), AM) }

{ text: /(\d\d?)(:?(\d\d))(:\d\d)?p\.?m\.?/ => TemporalCompose(INTERSECT, IsoTime($1,$3,$4), PM) }

{ text: /(\d\d?)a\.?m\.?/ => TemporalCompose(INTERSECT, IsoTime($1,"0",NIL), AM) }

{ text: /(\d\d?)p\.?m\.?/ => TemporalCompose(INTERSECT, IsoTime($1,"0",NIL), PM) }

{ ( /the/ /hour/ /of/ ([ $INT & { numcompvalue<=24 } ]) )=> IsoTime($1[0].numcompvalue, 0, NIL) }

{ ( (?: /the/ /hour/ /of/?)? ([ $INT & { numcompvalue<=24 } ]) /o'?clock/ ) => IsoTime($1[0].numcompvalue, 0, NIL) }

# Year

{ ( /the/? /year/ ($POSSIBLE\_YEAR) )

=> :case {

$1[0].word =~ /'.\*/ => IsoDate( GetTag($1[0], "YEAR"), NIL, NIL ),

:else => IsoDate( GetTag($1[0], "YEAR"), NIL, NIL, GetTag($1[0], "YEAR\_ERA"), TRUE)

}

}

{ ( ($POSSIBLE\_YEAR) & [ { ner::IS\_NIL } | { ner:DATE } | { ner:O } | { ner:NUMBER } ]+ )

=> :case {

$1[0].word =~ /'.\*/ => IsoDate( GetTag($1[0], "YEAR"), NIL, NIL ),

:else => IsoDate( GetTag($1[0], "YEAR"), NIL, NIL, GetTag($1[0], "YEAR\_ERA"), TRUE)

}

}

{ ( /the/ ($TIMEOFDAY) ) => $1[0].temporal.value }

{ ( /good/ /morning|evening|day|afternoon|night/ ) => NON\_TEMPORAL }

########################################################################################################################

# Compositional rules

ENV.defaults["ruleType"] = "composite"

{ name: "temporal-composite-6b",

priority: 20,

pattern: ( ( $REL\_MOD ) ( [ $hasTemporal & !{ temporal::IS\_TIMEX\_SET } ] ) ),

result: RelativeTime( GetTag($1[0], "TemporalOp"), $2[0].temporal.value )

}

########################################################################################################################

# Composite Duration rules

ENV.defaults["priority"] = 10

ENV.defaults["stage"] = 2

# Duration (start, end, unit, range\_start, range\_end)

# Duration rules

# i.e. "the past twenty four years"

{ pattern: ( /the/ /past|last|previous/ (?: ($NUM) /to|-/ )? ($NUM)? ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration($1,$2,$3,TIME\_UNKNOWN,TIME\_REF)

}

{ pattern: ( /the/ /next|following/ (?: ($NUM) /to|-/ )? ($NUM)? ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration($1,$2,$3,TIME\_REF,TIME\_UNKNOWN)

}

# i.e. "another 3 years", "another thirteen months"

{ pattern: ( /another/ (?: ($NUM) /to|-/ )? ($NUM)? ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration($1, $2, $3, TIME\_REF, TIME\_UNKNOWN) }

# i.e. "the 2 months following the crash", "for ten days before leaving"

# TODO: NEED TO FIX THIS, right now it doesn't include "the crash" or "leaving"

# ...need to be able to recognize NPs and VPs using POS tags

{ pattern: ( /the/ (?: ($NUM) /to|-/ )? ($NUM) ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration($1, $2, $3) }

# i.e. "the first 9 months of 1997"

{ pattern: ( /the/ /first|initial|last|final|latest/ (?: ($NUM) /to|-/ )? ($NUM)? ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration( $1, $2, $3 ) }

{ pattern: ( /the/ /first|initial|last|final|latest/ /half/ /of/ ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration( $1, $2, $3 ) }

# i.e. "the fifth straight year", "the third straight month in a row", "the ninth day consecutively"

# i.e. "the eighth consecutive day in a row"

# i.e. "the twenty ninth day straight"

{ pattern: ( /the/ ($NUM\_ORD) /straight|consecutive/ ([ { temporal::IS\_TIME\_UNIT } ]) (?: /in/ /a/ /row/ | /consecutively/ )? ),

result: Duration (NIL, $1, $2, TIME\_UNKNOWN, TIME\_REF) }

{ pattern: ( /the/ ($NUM\_ORD) /straight|consecutive/? ([ { temporal::IS\_TIME\_UNIT } ]) (?: /in/ /a/ /row/ | /consecutively/ ) ),

result: Duration (NIL, $1, $2, TIME\_UNKNOWN, TIME\_REF) }

# hundreds of years

{ pattern: ( (/(ten|hundred|thousand|million|billion|trillion)s/) /of/ ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration ( NIL, $1, $2) }

# i.e. "recent weeks", "several days"

{ pattern: ( (/recent|several/) /-/? ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: InexactDuration( Duration(NIL, NIL, $2) ) }

# i.e. 3-months old, "four years", "four minutes"

{ pattern: ( ($NUM) /to|-/ ($NUM) [ "-" ]? ([ { temporal::IS\_TIME\_UNIT } ]) (?: [ "-" ]? /old/ )? ),

result: Duration( $1, $2, $3) }

{ pattern: ( ($NUM) [ "-" ]? ([ { temporal::IS\_TIME\_UNIT } ]) (?: [ "-" ]? /old/ )? ),

result: Duration( NIL, $1, $2 ) }

# i.e. "a decade", "a few decades", NOT "a few hundred decades"

{ pattern: ( (?: /the/ /past|next|following|coming|last|first|final/ | /a|an/ )? (/couple/ /of/? ) ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration ( Duration( NIL, NIL, $2 ), 2 ) }

{ pattern: ( (?: /the/ /past|next|following|coming|last|first|final/ /half/ /of/ ) ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: InexactDuration ( Duration( NIL, NIL, $2 ) ) }

{ pattern: ( (?: /the/ /past|next|following|coming|last|first|final/ | /a|an/ )? (/few/) ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: InexactDuration ( Duration( NIL, NIL, $2 ) ) }

{ pattern: ( /the/ [ { tag:JJ } ]? ([ { temporal::IS\_TIME\_UNIT } & { word:/.\*s/ } ]) ),

priority: -1,

result: InexactDuration( $1[0].temporal.value ) }

{ pattern: ( /a|an/ ([ { temporal::IS\_TIME\_UNIT } ]) ),

result: Duration( $1[0].temporal.value, 1) }

######################################################################################################################

# Compositional date rules

ENV.defaults["priority"] = 0

{ ( ($FREQ\_MOD|/the/)? ($NUM\_ORD) ([ { temporal::IS\_TIME\_UNIT } | { temporal::DAYOFWEEK } ]) ) =>

:case{

# Only attach ordinal to time if not prefixed by frequency modifier (e.g. every)

($1 == NIL || $1 =~ ( /the/ ) ) => OrdinalTime($3[0].temporal.value, $2[0].numcompvalue),

# Return NIL otherwise - we have other patterns that handle sets (every 3rd months) later

:else => NIL

}

}

{ name: "composite-date-expression-1a",

priority: 1,

pattern: ( (/every/ $NUM\_ORD) (?$month [ { temporal::MONTH }]) ),

result: MakePeriodicTemporalSet(

$month[0].temporal,

GetTag($1[0], "PTS.quant"), GetTag($1[0], "PTS.multiple") ) }

{ name: "composite-date-expression-1",

priority: 1,

pattern: ( ( /the/? (?$day $NUM\_ORD & $INT1TO31) /of/? | (?$day /\d\d?/ & $INT1TO31) )?

(?$month [ { temporal::MONTH } ])

(?$day $NUM\_ORD|/\d\d?/ & $INT1TO31)?

(?: /of|,/? (?$year $POSSIBLE\_YEAR))? ),

result: TemporalCompose(INTERSECT,

$year[0].temporal,

IsoDate(NIL, $month[0].temporal.value.month, $day[0].numcompvalue))

}

{ name: "composite-date-expression-1b",

pattern: ( /the/? /ides/ /of|in/? (?$month [ { temporal::MONTH } ]) (?: /of|,/? (?$year $POSSIBLE\_YEAR))? ),

result: TemporalCompose(INTERSECT,

$year[0].temporal,

IsoDate(NIL,

$month[0].temporal.value.month,

:case {

$month[0].temporal.value.month == 3 => 15,

$month[0].temporal.value.month == 5 => 15,

$month[0].temporal.value.month == 7 => 15,

$month[0].temporal.value.month == 10 => 15,

:else => 13

}

))

}

{ name: "composite-date-expression-1c",

pattern: ( /the/? /nones/ /of|in/? (?$month [ { temporal::MONTH } ]) (?: /of|,/? (?$year $POSSIBLE\_YEAR))? ),

result: IsoDate(

GetTag($year[0].numtokens[0], "YEAR"),

$month[0].temporal.value.month,

:case {

$month[0].temporal.value.month == 3 => 7,

$month[0].temporal.value.month == 5 => 7,

$month[0].temporal.value.month == 7 => 7,

$month[0].temporal.value.month == 10 => 7,

:else => 5

}

)

}

{ name: "composite-date-expression-2",

pattern: ( /the/? (?$mod /beginning|start|middle|mid-?|end/ ) /of|in/? (?$date [ { temporal::IS\_TIMEX\_DATE } ]) ),

result: TemporalCompose(ADD\_MODIFIER, $date[0].temporal, GetTag($mod[0], "Modifier") ) }

{ name: "composite-date-expression-2a1",

pattern: ( /the/? (?$mod /first/ /half/) /of/ (?$date [ { temporal::IS\_TIMEX\_DATE } ]) ),

result: TemporalCompose(ADD\_MODIFIER, $date[0].temporal, "EARLY" ) }

{ name: "composite-date-expression-2a2",

pattern: ( /the/? (?$mod /second|last|latter/ /half/) /of/ (?$date [ { temporal::IS\_TIMEX\_DATE } ]) ),

result: TemporalCompose(ADD\_MODIFIER, $date[0].temporal, "LATE" ) }

{ name: "composite-date-expression-2b",

pattern: ( /the/? (?$date [ { temporal::IS\_TIMEX\_DATE } ]) (/'/ /s/ | /'s/ )? (?$mod /beginning|end/) ),

result: TemporalCompose(ADD\_MODIFIER, $date[0].temporal, GetTag($mod[0], "Modifier") ) }

{ name: "composite-date-expression-3",

pattern: ( /the/? (?$weeknum ($NUM\_ORD|last)) (?$week /week(end)?/ ) /of|in/? [ { temporal::IS\_TIMEX\_DATE } ] ),

result: TemporalCompose(

IN,

$0[-1].temporal,

TemporalCompose(

CREATE,

$week[0].temporal,

:case {

$weeknum =~ (/last/) => -1,

:else => $weeknum[0].numcompvalue

} ))

}

{ name: "composite-date-expression-3b",

pattern: ( /the/? (?$week /week(end)?/ ) /of|in/? [ { temporal::IS\_TIMEX\_DATE } ] ),

result: TemporalCompose(

INTERSECT,

$0[-1].temporal,

$week[0].temporal)

}

{ name: "composite-date-expression-3c",

pattern: ( ( [ { temporal::DAYOFWEEK } ] ) /the/? (?$day $NUM\_ORD) ),

result: TemporalCompose(

INTERSECT,

$1[0].temporal,

IsoDate(NIL, NIL, $day[0].numcompvalue))

}

{ name: "composite-date-expression-6",

pattern: ( ([ { temporal::IS\_TIMEX\_DATE } ]) (morning|afternoon|evening|night) ),

result: TemporalCompose(INTERSECT, $1[0].temporal, $2[0].temporal)

}

{ name: "composite-date-expression-7a",

pattern: ( (?: /the/? /day/ (/before/|/prior/ /to/) ([ { temporal::IS\_TIMEX\_DATE } ]) ) ),

result: TemporalCompose(

PLUS,

$2[0].temporal,

TemporalCompose(

MULTIPLY,

DAY,

-1))

}

{ name: "composite-date-expression-7b",

pattern: ( (?: /the/? /day/ (/after/) ([ { temporal::IS\_TIMEX\_DATE } ]) ) ),

result: TemporalCompose(

PLUS,

$2[0].temporal,

DAY)

}

{ name: "composite-date-expression-8",

pattern: ( /the/ [ { tag:JJ } ]\* ([ { temporal::IS\_TIME\_UNIT }

& !{ word:/.\*s/ } ] )),

result: RelativeTime( THIS, $1[0].temporal.value )

}

########################################################################################################################

# Composite time expressions

{ name: "composite-time-expression-1a",

active: TRUE,

pattern: ( (?: (?$time [ { temporal::IS\_TIMEX\_TIME } ]) | (?$hour [ $INT & { numcompvalue<=24 } ]))

(?$context /in/ /the/ /morning/ | /a\.?m\.?/)

),

result: :case {

$time => TemporalCompose(INTERSECT, $time[0].temporal.value, AM),

$hour[0].numcompvalue == 12 => IsoTime(0, 0, NIL),

:else => IsoTime($hour[0].numcompvalue, 0, NIL)

}

}

{ name: "composite-time-expression-1b",

active: TRUE,

pattern: ( (?: (?$time [ { temporal::IS\_TIMEX\_TIME } ]) | (?$hour [ $INT & { numcompvalue<=24 } ]))

(?$context /in/ /the/ /afternoon|evening/| /at/ /night/| /p\.?m\.?/)

),

result: :case {

$time => TemporalCompose(INTERSECT, $time[0].temporal.value, PM),

$hour[0].numcompvalue < 12 => IsoTime(Add($hour[0].numcompvalue, 12), 0, NIL),

( ($hour[0].numcompvalue == 12) && ($context =~ ( []\* /evening|night/)) )

=> TemporalCompose(OFFSET\_EXACT, IsoTime(0, 0, NIL), DAY),

:else => IsoTime($hour[0].numcompvalue, 0, NIL)

}

}

{ name: "composite-time-expression-1c",

active: TRUE,

pattern: ( (?: (?$time [ { temporal::IS\_TIMEX\_TIME } ]) | (?$hour [ $INT & { numcompvalue==12 } ]))

(?$context /midnight/)

),

result: :case {

$time[0].temporal.value.hour == 12 => MIDNIGHT,

$hour[0].numcompvalue == 12 => MIDNIGHT,

:else => NIL

}

}

{ name: "composite-time-expression-2",

pattern: ( (?$minute /a/? /quarter/ | /a/? /half/ | [ $INT & { numcompvalue<=60 } ] /minutes?/? )

(?$rel /past|after|before|to|until/)

(?: (?$time [ { temporal::IS\_TIMEX\_TIME } ]) | (?$hour [ $INT & { numcompvalue<=24 } ]))

),

result: TemporalCompose(

:case {

$rel[0].word =~ /past|after/ => PLUS,

:else => MINUS

},

:case {

$time => $time[0].temporal,

:else => IsoTime($hour[0].numcompvalue, 0, NIL)

},

Duration(

MINUTE,

:case {

$minute =~ ( /a/? /quarter/ ) => 15,

$minute =~ ( /a/? /half/ ) => 30,

:else => $0[0].numcompvalue

}

)

) }

{ pattern: ( ( /\d\d\d\d/ | $NUM ) [ {tag:/RB/} ] [ {tag:/JJ/} ]+ [ {tag:/NNS/} & !($hasTemporal) ] ),

result: NON\_TEMPORAL,

priority: -1 }

{ pattern: ( ( /\d\d\d\d/ | $NUM ) [ {tag:/JJ/} ]\* [ {tag:/NNS/} & !($hasTemporal) ] ),

result: NON\_TEMPORAL,

priority: -1 }

########################################################################################################################

# General compositional rules

ENV.defaults["stage"] = 3

{ name: "temporal-composite-timezone1",

pattern: ( (?$time [ { temporal::IS\_TIMEX\_TIME } ]) (?$timezone [ {{ tags["TIMEZONE"] }} ]) ),

result: TemporalCompose(INTERSECT, $time[0].temporal, $timezone[0].temporal)

}

{ name: "temporal-composite-timezone2",

pattern: ( (?$time [ { temporal::IS\_TIMEX\_TIME } ]) "-LRB-" (?$timezone [ {{ tags["TIMEZONE"] }} ]) "-RRB-" ),

result: TemporalCompose(INTERSECT, $time[0].temporal, $timezone[0].temporal)

}

{ name: "temporal-composite-1",

pattern: ( /the/?

(( [ $hasTemporal ] ) /,|of|in/? ( [ { temporal::IS\_TIMEX\_DATE } | { temporal::IS\_TIMEX\_TIME } ] ) |

( [ { temporal::IS\_TIMEX\_DATE } ] ) /at/ ( [ { temporal::IS\_TIMEX\_TIME } ] ) |

( [ { temporal::IS\_TIMEX\_TIME } | { temporal::IS\_TIMEX\_DURATION } ] ) /on/ ( [ { temporal::IS\_TIMEX\_DATE } ] | [ { temporal::IS\_TIMEX\_SET } ]) |

( [ { temporal::IS\_TIMEX\_DATE } | { temporal::IS\_TIMEX\_TIME } ] ) (/'s/ | /'/ /s/) ( [ $hasTemporal ] )) ),

result: TemporalCompose(INTERSECT, $1[0].temporal, $1[-1].temporal)

}

{ name: "temporal-composite-2",

pattern: ( ( [ { temporal::IS\_TIMEX\_DATE } | { temporal::IS\_TIMEX\_TIME } ] ) (/today|tonight/) ),

result: $0[0].temporal.value

}

{ name: "temporal-composite-3",

pattern: ( ( [ { temporal::IS\_TIMEX\_DURATION } ] ) (/before|from|since|after/ | /prior/ /to/)

( [ ({ temporal::IS\_TIMEX\_TIME } | { temporal::IS\_TIMEX\_DATE }) ] ) ),

result: TemporalCompose(

OFFSET, $0[-1].temporal,

TemporalCompose(

MULTIPLY,

$0[0].temporal,

:case {

$2 =~ (/before/|/prior/ /to/) => -1,

:else => 1

} ))

}

# expand: timex later|earlier|late => one timex

{ name: "temporal-composite-4",

pattern: ( ( [ { temporal::IS\_TIMEX\_DURATION } ] ) (/earlier|later|ago|hence/ | /from/ /now/) ),

result: TemporalCompose(OFFSET, TIME\_REF,

TemporalCompose( MULTIPLY, $0[0].temporal,

:case {

$2 =~ (/earlier/|/ago/) => -1,

:else => 1

}))

}

# expand: timex later|earlier|late => one timex

{ name: "temporal-composite-5",

pattern: ( ( [ $hasTemporal & !{ temporal::IS\_TIMEX\_DURATION } ] )

(/before|earlier|later|late|ago|hence/ | /from/ /now/) ),

result: $0[0].temporal.value }

{ name: "temporal-composite-6a",

pattern: ( /the/? ( $EARLY\_LATE\_MOD ) ( [ $hasTemporal & !{ temporal::IS\_TIMEX\_SET } ] ) ),

result: TemporalCompose( ADD\_MODIFIER, $2[0].temporal.value, GetTag($1[0], "Modifier") )

}

{ name: "temporal-composite-6b",

priority: 4,

pattern: ( ( $REL\_MOD ) ( [ $hasTemporal & !{ temporal::IS\_TIMEX\_SET } ] ) ),

result: RelativeTime( GetTag($1[0], "TemporalOp"), $2[0].temporal.value )

}

{ name: "temporal-composite-6b1",

priority: 4,

pattern: ( ( /no/ /more/ /than/ | /at/ /most/ | /up/ /to/ )

( [ { temporal::IS\_TIMEX\_DURATION } & !{{ temporal.value.mod }} ] ) ),

result: TemporalCompose( ADD\_MODIFIER, $0[-1].temporal.value, "EQUAL\_OR\_LESS" )

}

{ name: "temporal-composite-6b2",

priority: 4,

pattern: ( ( /more/ /than/ )

( [ { temporal::IS\_TIMEX\_DURATION } & !{{ temporal.value.mod }} ] ) ),

result: TemporalCompose( ADD\_MODIFIER, $0[-1].temporal.value, "MORE\_THAN" )

}

{ name: "temporal-composite-6b3",

priority: 4,

pattern: ( ( /no/ /less/ /than/ | /at/ /least/ )

( [ { temporal::IS\_TIMEX\_DURATION } & !{{ temporal.value.mod }} ] ) ),

result: TemporalCompose( ADD\_MODIFIER, $0[-1].temporal.value, "EQUAL\_OR\_MORE" )

}

{ name: "temporal-composite-6b4",

priority: 4,

pattern: ( ( /less/ /than/ )

( [ { temporal::IS\_TIMEX\_DURATION } & !{{ temporal.value.mod }} ] ) ),

result: TemporalCompose( ADD\_MODIFIER, $0[-1].temporal.value, "LESS\_THAN" )

}

# expand: (the|this|about|nearly|early|later|earlier|late) timex => one timex

# expand: more than| up to| less than timex => one timex

{ name: "temporal-composite-6c",

pattern: ( ( /this|about|nearly|early|later|earlier|late/ )

( [ $hasTemporal & !{ temporal::IS\_TIMEX\_SET } ] ) ),

result: $0[-1].temporal.value

}

{ name: "temporal-composite-7a",

pattern: ( /every/ ( [ $hasTemporal & !{ temporal::IS\_TIMEX\_SET } ] ) ),

result: MakePeriodicTemporalSet($1[0].temporal, "every", 1 )

}

{ name: "temporal-composite-7b",

# pattern: ( ( $FREQ\_MOD ) ( [ $hasTemporal & !{ temporal::IS\_TIMEX\_SET } ] ) ),

pattern: ( ( $FREQ\_MOD ) ( [ $hasTemporal ] ) ),

result: MakePeriodicTemporalSet($2[0].temporal, GetTag($1[0], "PTS.quant"), GetTag($1[0], "PTS.multiple") )

}

{ name: "temporal-composite-8:ranges",

active: options."markTimeRanges",

pattern: ( /from/? ( [ { temporal::IS\_TIMEX\_TIME } | { temporal::IS\_TIMEX\_DATE } ] ) /to|-/ ( [ { temporal::IS\_TIMEX\_TIME } | { temporal::IS\_TIMEX\_DATE } ] ) ),

result: TimeRange( $1[0].temporal.value, $2[0].temporal.value ) }

{ name: "temporal-composite-9",

pattern: ( [{ temporal::IS\_TIMEX\_TIME }] (?: /sharp/|/exactly/|/precisely/|/on/ /the/ /dot/) ),

result: $0[0].temporal.value }

########################################################################################################################

ENV.defaults["stage"] = 4

ENV.defaults["ruleType"] = "tokens"

# Vague times

{ ( /the/ /past/ | /recently/ ) => TIME\_PAST }

{ pattern: ( /at/ /the/ (/time/) ), matchedExpressionGroup: 1, result: TIME\_PAST }

{ ( /past|once|medieval|previously/ ) => TIME\_PAST }

{ ( /present|current|currently/ | /right/? /now/ ) => TIME\_PRESENT }

{ ( /the/? /near/? /future/ ) => TIME\_FUTURE }

# Final rules to determine how to resolve date

ENV.defaults["ruleType"] = "composite"

ENV.defaults["stage.limitIters"] = 1

{ pattern: ( [ { temporal::IS\_TIMEX\_DURATION } & {{ tokens =~ ( /.\*s/ ) }} ] ),

result: InexactDuration( $0[0].temporal.value ) }

{ pattern: ( [ { tag:/VBD/ } | /have/ ] []{0,2} [ $hasTemporal ] ),

action: VTag( $0[-1].temporal.value, "resolveTo", RESOLVE\_TO\_PAST )

}

{ pattern: ( [ $hasTemporal ] []{0,2} [ { tag:/VBD/ } | /have/ ] ),

action: VTag( $0[0].temporal.value, "resolveTo", RESOLVE\_TO\_PAST )

}

{ pattern: ( (/would/ | /could/ | /should/ | /will/ | /going/ /to/ | /'/ /ll/ | /'ll/ )

[]{0,2} [ $hasTemporal ]

),

action: VTag( $0[-1].temporal.value, "resolveTo", RESOLVE\_TO\_FUTURE )

}

{ pattern: ( [ $hasTemporal ] []{0,2}

(/would/ | /could/ | /should/ | /will/ | /going/ /to/ | /'/ /ll/ | /'ll/ ) ),

action: VTag( $0[0].temporal.value, "resolveTo", RESOLVE\_TO\_FUTURE )

}

########################################################################################################################

# Final filtering rules

ENV.defaults["ruleType"] = "filter"

ENV.defaultTokensAnnotationKey = tokens

{ pattern: ( $NUM /to/ $NUM) }

{ pattern: ( /(quarter|sun)s?/ ) }

{ pattern: ( [ { word:/(fall|spring|second|march|may|sat|sun|min)s?/ } & !{ tag:/NN.\*/ } ] ) }

{ pattern: ( /the/ [ { word:/second/ } & !{ tag:/NN.\*/ } ] ) }

{ pattern: ( [ { word:/((twenty|thirty|forty|fifty|sixty|seventy|eighty|ninety)-)?second/ } ] ) }

{ pattern: ( [ {{ temporal.value == NON\_TEMPORAL }} ] ), over: NIL }

# Reject anything that is just a timezone

{ pattern: ( [ {{ tags["TIMEZONE"] }} ] ), over: NIL }