Appendix C: Grammar

8.1 Notes about the parsing rules

Recall from the paper that we specify our grammar using rules of the following form: $\alpha_1 ... \alpha_n \rightarrow c[\beta]$.

The β can be exactly the target derivation, or a semantic function (such as IdentityFn, SelectFn, sketch.UnarySketchFn etc.) that is applied to produce target derivation with arguments.

There are two types of arguments:

- arg:i represents selecting the i^{th} position from the matched $\alpha_1 ... \alpha_n$ and passing it to the function. For example, the rule ($Skip $SKETCH $\rightarrow$ $ROOT [SelectFn arg:1]) selects the first category from the source sequence, which is the $SKETCH (Notes that Sempre starts indexing from 0).
- val:n represents passing the integer value n to the function. For example, the rule ($CC1 $MARKER_ONLY $\rightarrow$ $PROGRAM[sketch.RepeatatleastFn arg:0, val:1]) passes the first category in the matched source sequence $MARKER_ONLY and the integer value 1 to the semantic function sketch.RepeatatleastFn.

8.2 Compositional Rules

Root
$Sketch → $ROOT [IdentityFn arg:0]
$Skip $SKETCH → $ROOT [SelectFn arg:1]

Skip tokens rule
$LEMMA_PHRASE → $Skip [ConstantFn arg:null]

Parse a number
$LEMMA_PHRASE → $INT1 [NumberFn val:NUMBER]

Parse a character class or constant to a regex
$CC → $PROGRAM [IdentifyFn arg:0]
$CONST → $PROGRAM [IdentifyFn arg:0]

Hierarchical sketch parse rules
$LIST_PROGRAM → $SKETCH [sketch.UnarySketchFn arg:0]
$PROGRAM → $LIST_PROGRAM [IdentityFn arg:0]
$PROGRAM $LIST_PROGRAM → $LIST_PROGRAM [sketch.SketchJoinFn arg:0]

Operator: NotContain
$MARKER_NOTCONTAIN $SKETCH → $SKETCH [sketch.NotContainFn arg:1]
$MARKER_NOTCONTAIN $PROGRAM → $PROGRAM [sketch.NotContainFn arg:1]

Operator: Not
$MARKER_NOT $SKETCH → $SKETCH [sketch.NotFn arg:1]
$MARKER_NOT $PROGRAM → $PROGRAM [sketch.NotFn arg:1]
$MARKER_NON1 $CONST → $PROGRAM [sketch.NotccFn arg:1]

Operator: Optional
$MARKER_NOT $CC → $PROGRAM [sketch.OptionalFn arg:1]

Operator: StartWith, EndWith
$MARKER_STARTWITH $PROGRAM → $PROGRAM [sketch.StartwithFn arg:1]
$MARKER_ENDWITH $PROGRAM → $PROGRAM [sketch.EndwithFn arg:1]
$PROGRAM $MARKER_ATEND → $PROGRAM [sketch.EndwithFn arg:0]

Operator: Concat
$PROGRAM $MARKER_CONCAT $PROGRAM → $PROGRAM [sketch.ConcatFn arg:0, arg:2]
$PROGRAM $MARKER_CONCAT $SKETCH → $SKETCH [sketch.ConcatFn arg:0, arg:2]
$SKETCH $MARKER_CONCAT $PROGRAM → $SKETCH [sketch.ConcatFn arg:0, arg:2]
$SKETCH $MARKER_CONCAT $SKETCH → $SKETCH [sketch.ConcatFn arg:0, arg:2]
$PROGRAM $MARKER_FOLLOW $PROGRAM → $PROGRAM [sketch.ConcatFn arg:2, arg:0]
$PROGRAM $MARKER_FOLLOW $SKETCH → $SKETCH [sketch.ConcatFn arg:2, arg:0]
$SKETCH $MARKER_FOLLOW $PROGRAM → $SKETCH [sketch.ConcatFn arg:2, arg:0]
$SKETCH $MARKER_FOLLOW $SKETCH → $SKETCH [sketch.ConcatFn arg:2, arg:0]

Operator: Repeat

$INT $CC → $PROGRAM [sketch.RepeatFn arg:1, arg:0]
$CC $MARKER_LENGTH $INT → $PROGRAM [sketch.RepeatFn arg:0, arg:2]
$MARKER_LENGTH $INT $CC → $PROGRAM [sketch.RepeatFn arg:2, arg:1]
$INT1 $MARKER_OR1 $INT1 $CC → $PROGRAM [sketch.RepeatAOrBFn arg:3, arg:0, arg:2]

Operator: RepeatAtLeast
$MARKER_ONLY1 $CC → $PROGRAM [sketch.RepeatatleastFn arg:1, val:1]
$CC1 $MARKER_ONLY → $PROGRAM [sketch.RepeatatleastFn arg:0, val:1]
$INT1 $MARKER_ORMORE1 $CC → $PROGRAM [sketch.RepeatatleastFn arg:2, arg:0]
$PROGRAM $INT1 $MARKER_ORMORE1 → $PROGRAM [sketch.RepeatatleastFn arg:0, arg:1]

Operator: RepeatRange
$MARKER_ATMAX1 $INT $CC → $PROGRAM [sketch.RepeatrangeFn arg:2, val:1, arg:1]
$MARKER_ATMAX $INT $CC → $PROGRAM [sketch.RepeatrangeFn arg:2, val:1, arg:1]
$INT $CC → $PROGRAM [sketch.RepeatrangeFn arg:1, val:1, arg:0]

Extract constants
$CONST_SET → $CC1 [IdentityFn arg:0]
$CONST1 $CONST_SET → $CONST_SET [sketch.ConstUnionFn arg:0, arg:1]
$CONST1 $CONST1 → $CONST_SET [sketch.ConstUnionFn arg:0, arg:1]
$CONST1 $MAKRKER_CONSTSETUNION1 $CONST1 → $CONST_SET [sketch.ConstUnionFn arg:0, arg:2]
$CONST1 $MAKRKER_CONSTSETUNION1 $CONST_SET → $CONST_SET [sketch.ConstUnionFn arg:0, arg:2]
$CCPHRASE1 $MAKRKER_CONSTSETUNION1 $CONST1 → $CONST_SET [sketch.ConstUnionFn arg:0, arg:2]
$CCPHRASE1 $MAKRKER_CONSTSETUNION1 $CCPHRASE1 → $CONST_SET [sketch.ConstUnionFn arg:0, arg:2]
$CCPHRASE1 $MAKRKER_CONSTSETUNION1 $CONST_SET → $CONST_SET [sketch.ConstUnionFn arg:0, arg:2]
“ $PHRASE ” → $CONST1 [sketch.ConstFn arg:0]

“Separated/Split by”
$SKETCH $PROGRAM $MARKER_SEP → $SKETCH [sketch.SepFn arg:0,arg:1]
$PROGRAM $PROGRAM $MARKER_SEP → $PROGRAM [sketch.SepFn arg:0,arg:1]
$PROGRAM $MARKER_BETWEEN $SKETCH → $SKETCH [sketch.SepFn arg:2,arg:0]
$PROGRAM $MARKER_BETWEEN $PROGRAM → $PROGRAM [sketch.SepFn arg:2,arg:0]
$SKETCH $MARKER_SPLITBY $PROGRAM → $SKETCH [sketch.SepFn arg:0,arg:2]

“Decimal”
$PROGRAM $MARKER_DECIMAL $PROGRAM → $SKETCH [sketch.DecimalFn arg:0, arg:2]
$MARKER_DECIMAL $PROGRAM $PROGRAM → $SKETCH [sketch.DecimalFn arg:1, arg:2]
$PROGRAM $PROGRAM $MARKER_DECIMAL → $SKETCH [sketch.DecimalFn arg:0, arg:1]
$MARKER_DECIMALNUM → $SKETCH [sketch.DecimalFn]

Skip Rules: we present one example here as a skip rule that is required by Sempre to allow skipping tokens when matching compositional rules. These rules can be generated automatically and hence we don’t count these as part of the compositional rules.
$Skip optional → $CC [SelectFn arg:0]

Lexicon mapping rules: we present one example here that matches lexicons in the lexicon files to base-case target category to allow compositional rules build up on lexicons. These rules can be generated automatically and hence we don’t count these as part of the compositional rules.
$CCPHRASE1 → $CC1 [IdentityFn arg:0]

8.3 Lexical Rules

number → $CC [<num>]
numeric → $CC [<num>]
numeral → $CC [<num>]
decimal → $CC [<num>]
digit → $CC [<num>]
alphanumeric → $CC [<alphanum>]
hexadecimal → $CC [<hex>]
string → $CC [<any>]
character → $CC [<let>]
letter → $CC [<let>]
word → $CC [<let>]
alphabet → $CC [<let>]

lower case letter → $CC [<low>]
small letter → $CC [<low>]
upper case letter → $CC [<cap>]
capital letter → $CC [<cap>]
vowel → $CC [<vow>]
special character → $CC [<spec>]
special char → $CC [<spec>]
comma → $CONST [<,>]
colon → $CONST [<:>]
semicolon → $CONST [<;>]
space → $CONST [<space>]
underscore → $CONST [<_>]
dash → $CONST [<->]
percentage sign → $CONST [<%>]
percentage sign → $CONST [<%>]
not → $OP.NOT [op.not]
non → $OP.NON [op.non]
or → $OP.OR [op.or]
optional → $OP.OPTIONAL [op.optional]
not contain → $OP.NOTCONTAIN [op.notcontain]
not allow → $OP.NOTCONTAIN [op.notcontain]
or more → $OP.ORMORE [op.ormore]
or more time → $OP.ORMORE [op.ormore]
max → $OP.MAX [op.max]
decimal → $OP.DECIMAL [op.decimal]
double number → $OP.DECIMALNUM [op.decimalnum]
length → $OP.LENGTH [op.length]
, → $OP.CONSTSETUNION [op.constsetunion]
(, optional) or → $OP.CONSTSETUNION [op.constsetunion]
(, optional) and → $OP.CONSTSETUNION [op.constsetunion]
separate → $OP.SEP [op.sep]
delimit → $OP.SEP [op.sep]
between → $OP.BETWEEN [op.between]
separated → $OP.BETWEEN [op.between]
split by → $OP.SPLITBY [op.splitby]
divide by → $OP.SPLITBY [op.splitby]
end with → $OP.ENDWITH [op.endwith]
finish with → $OP.ENDWITH [op.endwith]
end in → $OP.ENDWITH [op.endwith]
terminate → $OP.ENDWITH [op.endwith]
at end → $OP.ATEND [op.atend]
start with → $OP.STARTWITH [op.startwith]
start in → $OP.STARTWITH [op.startwith]
at the begin → $OP.STARTWITH [op.startwith]
before → $OP.CONCAT [op.concat]
follow by → $OP.CONCAT [op.concat]
next → $OP.CONCAT [op.concat]
then → $OP.CONCAT [op.concat]
prior to → $OP.CONCAT [op.concat]
precede → $OP.CONCAT [op.concat]
after → $OP.FOLLOW [op.follow]
bulletpoint → $OP.FOLLOW [op.follow]
up to → $OP.ATMAX [op.atmax]
at max → $OP.ATMAX [op.atmax]
only → $OP.ONLY [op.only]