From Programs to Context-Free Grammars

1. The rules we used to define programs make up a context-free grammar

A Context-Free Grammar is a tuple \( (C, X, S, R) \), where:

- \( C \) is a set of category symbols
- \( X \subseteq C \) is a set of terminal symbols (e.g. if, :, ...)
- \( S \subseteq C \) is a set of start symbols (e.g. \( \langle \text{program} \rangle \))
- \( R \subseteq C \times C \times C \) is a set of rewrite rules of the form \( c \rightarrow c' c'' \) (or \( \langle c, c', c'' \rangle \))

The symbol to the left of each arrow may be called the ‘parent’
The symbols to the right of each arrow may be called the ‘children’
Grammars allowing only two children per rule are called ‘binary-branching’

For example:

\[
G_{\text{cattoy}} = \langle \{ S, NP, VP, \text{the, cat, hits, toy} \}, \{ \text{the, cat, hits, toy} \}, \{ S \}, \{ S \rightarrow NP VP, NP \rightarrow \text{the cat}, NP \rightarrow \text{the toy}, VP \rightarrow \text{hits NP} \} \rangle
\]

2. Languages accepted by a CFG

For any CFG \( G = (C, X, S, R) \):

\[
L(G) = \{ x_1 \ldots x_T \mid \exists c \in S \text{ s.t. } c \xrightarrow{G} x_1 \ldots x_T \}
\]

where \( c \) yields \( x_i \ldots x_j \) if and only if there is a tree \( \tau \) w. root \( c \), all leaves in \( x_i \ldots x_j \), branches in \( R \)
(defined as sets of constituents \( \langle c, i, j \rangle \) of category \( c \) spanning input \( i \) to \( j \)):

\[
c \xrightarrow{G} x_i \ldots x_j \text{ iff } \bigvee_{\tau \text{ w. root } \langle c, i, j \rangle} \bigwedge_{\langle c', i', j' \rangle \in \tau} \left\{ \begin{array}{ll}
\text{if } i' = j' : & \{ \text{if } c' = x_{i'} : \text{True} \\
\text{if } c' \neq x_{i'} : \text{False} \}
\end{array} \right.
\]

For example, ‘the cat hits the toy’ is in \( L(G_{\text{cattoy}}) \) because this tree \( \tau \) exists:

```
S
   NP  VP
      the cat hits NP
           the toy
```

with constituents:

\[
\tau = \{ \langle S, 1, 5 \rangle, \langle NP, 1, 2 \rangle, \langle \text{the}, 1, 1 \rangle, \langle \text{cat}, 2, 2 \rangle, \langle VP, 3, 5 \rangle, \langle \text{hits}, 3, 3 \rangle, \langle NP, 4, 5 \rangle, \langle \text{the}, 4, 4 \rangle, \langle \text{toy}, 5, 5 \rangle \}
\]
We can now move one branch conjunct outside the disjunction over trees (distributive axiom):

\[ c \overset{*}{\rightarrow}_{G} x_{i}\ldots x_{j} \iff \begin{cases} \text{if } i = j : & \begin{cases} \text{if } c = x_{i} : & \text{True} \\ \text{if } c \neq x_{i} : & \text{False} \end{cases} \\ \text{if } i < j : & \bigvee_{k,d,e} R(c \rightarrow d \ e) \wedge \left( \bigvee_{\tau'} \wedge \left\{ \ldots \right\} \right) \wedge \left( \bigvee_{\tau''} \wedge \left\{ \ldots \right\} \right) \end{cases} \]

and identify parenthesized terms as recursive instances of \( c \overset{*}{\rightarrow}_{G} \):

\[ c \overset{*}{\rightarrow}_{G} x_{i}\ldots x_{j} \iff \begin{cases} \text{if } i = j : & \begin{cases} \text{if } c = x_{i} : & \text{True} \\ \text{if } c \neq x_{i} : & \text{False} \end{cases} \\ \text{if } i < j : & \bigvee_{k,d,e} R(c \rightarrow d \ e) \wedge \left( d \overset{*}{\rightarrow}_{G} x_{i}\ldots x_{k} \right) \wedge \left( e \overset{*}{\rightarrow}_{G} x_{k+1}\ldots x_{j} \right) \end{cases} \]

More simply:

\[ c \overset{*}{\rightarrow}_{G} x_{i}\ldots x_{j} \iff \begin{cases} \text{if } i = j : & c = x_{i} \\ \text{if } i < j : & \exists k, d, e \text{ s.t. } c \rightarrow d \ e \in R \text{ and } d \overset{*}{\rightarrow}_{G} x_{i}\ldots x_{k} \text{ and } e \overset{*}{\rightarrow}_{G} x_{k+1}\ldots x_{j} \end{cases} \]

And the languages recognizable by a CFG are:

\[ \mathcal{L}(CFG) = \{ L(G) \mid CFG \ G \} \]

3. \( \mathcal{L}(CFG) \subseteq \mathcal{L}(CFG_{\leq 2 \text{ children per branch}}) \), so \( \mathcal{L}(CFG) = \mathcal{L}(CFG_{\leq 2 \text{ children per branch}}) \)

CFGs which have rules \( R \) containing more than two children can be reduced to binary \( R' \):

\( R' = \{ c \rightarrow c_{0} c_{1}\ldots c_{B} \mid c \rightarrow c_{0} \ldots c_{B} \in R \} \)

\( \cup \{ c_{b_{1}\ldots c_{B}} \rightarrow c_{b} c_{b+1}\ldots c_{B} \mid c \rightarrow c_{0} \ldots c_{B} \in R, \ 1 \leq b < B \} \)

\( C' = C \cup \{ c_{b_{1}\ldots c_{B}} \mid c \rightarrow c_{0} \ldots c_{B} \in R, \ 1 \leq b < B \} \)

For example:

\( VP \rightarrow \text{pick NP up,} \)

\( \ldots \)

would become:

\( VP \rightarrow \text{pick NP up,} \)

\( \text{NP up} \rightarrow \text{NP up,} \)

\( \ldots \)

4. \( \mathcal{L}(CFG_{\leq 2 \text{ children per branch}}) \subseteq \mathcal{L}(CFG_{2 \text{ children per branch}}) \), so \( \mathcal{L}(CFG) = \mathcal{L}(CFG_{2 \text{ children per branch}}) \)

CFGs which have rules \( R \) containing fewer than two children can be reduced to binary \( R' \):

(a) First, obtain reflexive transitive closure \( R^{(\leq C)} \) of unary rules in \( R \) (like \( \epsilon \)-transitions in FSA):

\[ R^{(0)} = \{ c \rightarrow c \mid c \in C \} \]

\[ R^{(k)} = \{ c \rightarrow c'' \mid c \rightarrow c' \in R^{(k-1)}, \ c' \rightarrow c'' \in R \} \]

(b) Then add these closure expansions to children of each binary expansion:

\( R' = \{ c \rightarrow d' e' \mid c \rightarrow d \ e \in R, \ d \rightarrow d' \in R^{(\leq C)}, \ e \rightarrow e' \in R^{(\leq C)} \} \)
(c) Then add these closure expansions to the set of start symbols:
\[ S' = S \cup \{ c' | c \in S, c \rightarrow c' \in R^{[\lceil C \rceil]} \} \]

For example, adding the following unary rule to our cat-toy grammar:

\[ \text{NP} \rightarrow \text{Meowy} \]

would result in the addition of the following binary rules:

\[ S \rightarrow \text{Meowy VP}, \quad \text{VP} \rightarrow \text{hits Meowy} \]

5. \( L(\text{RE}) \subset L(\text{CFG}) \)

We can write a CFG \( G(\rho) \) implementing any RE \( \rho \), concatenated to a start symbol:

first, remove epsilons from Kleene star:

\[ \rho^* = (\epsilon|\rho^+) \]
\[ (\epsilon|\rho_1)\rho_2 = (\rho_2|\rho_1\rho_2) \]
\[ \rho_1(\epsilon|\rho_2) = (\rho_1|\rho_1\rho_2) \]

then:

\[ R_{G(\rho_1\rho_2)} = R_{G(\rho_1)} \cup R_{G(\rho_2)} \cup \{ c_{\rho_1\rho_2} \rightarrow c_{\rho_1} c_{\rho_2} \} \]
\[ R_{G(\rho_1|\rho_2)} = R_{G(\rho_1)} \cup R_{G(\rho_2)} \cup \{ c_{\rho_1|\rho_2} \rightarrow c_{\rho_1} c_{\rho_2} \} \]
\[ R_{G(\rho_1^+)} = R_{G(\rho_1)} \cup \{ c_{\rho_1^+} \rightarrow c_{\rho_1} c_{\rho_1^+}, c_{\rho_1^+} \rightarrow c_{\rho_1} \} \]

6. \( L(\text{RE}) \subset L(\text{CFG}) \)

There is a language that cannot be recognized by a RE: \( a^n b^n \)

(proof by pumping lemma)

...which can be recognized by a CFG:

(proof by existance: \( \langle \{ S, a, b \}, \{ a, b \}, \{ S \}, \{ S \rightarrow a S b, S \rightarrow a b \} \rangle \))

7. Can use CFG for natural language: \( G_{\text{English}} = \langle C, X, S, R \rangle \) (generatively sound, not complete)

(a) Categories \( C \) defined from set of primitives \( U \) and set of type-constructing operators \( O \):

i. Primitive categories \( U \) — different ways to say the same thing:

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>finite / declarative sentence (e.g. ‘they knew it’)</td>
</tr>
<tr>
<td>I</td>
<td>infinitive clause (e.g. ‘them to know it’, as in ‘we expect . . . ’)</td>
</tr>
<tr>
<td>B</td>
<td>base-form clause (e.g. ‘them know it’, as in ‘we let . . . ’)</td>
</tr>
<tr>
<td>L</td>
<td>participle clause (e.g. ‘them known it’ — complete form not used)</td>
</tr>
<tr>
<td>A</td>
<td>adjectival/predicative/‘small’ clause (e.g. ‘them knowing it’, ‘it known’, as in ‘we consider . . . ’)</td>
</tr>
<tr>
<td>R</td>
<td>adverbial clause (e.g. ‘them knowingly’ — complete form not used)</td>
</tr>
<tr>
<td>G</td>
<td>gerund clause (e.g. ‘them knowing it’)</td>
</tr>
<tr>
<td>N</td>
<td>accus. noun phrase / nominal clause (e.g. ‘their knowledge of it’, ‘her’, ‘him’, ‘Kim’)</td>
</tr>
<tr>
<td>N1S</td>
<td>nom. 1st-pers. sing. noun phrase (e.g. ‘I’)</td>
</tr>
<tr>
<td>N1P</td>
<td>nom. 1st-pers. plur. noun phrase (e.g. ‘we’)</td>
</tr>
<tr>
<td>N2</td>
<td>nom. 2nd-pers. noun phrase (e.g. ‘you’)</td>
</tr>
</tbody>
</table>
N3S nom. 3rd-pers. sing. noun phrase (e.g. ‘their knowledge of it’, ‘she’, ‘he’, ‘Kim’)
N3P nom. 3rd-pers. plur. noun phrase (e.g. ‘they’, ‘the risks’)
D determiner (clause) (e.g. ‘their knowledge of it’s’, ‘the’, ‘their’)
O non-possessive genitive (clause) (e.g. ‘of their knowledge of it’)
C complemented clause (e.g. ‘that they know it’, as in ‘we think . . . ’)
E embedded infinitive clause (e.g. ‘for them to know it’, as in ‘we hope . . . ’)
Q (polar) question (e.g. ‘did they know it’)
S start symbol / sentence / utterance (e.g. ‘know it’, ‘yeah’, ‘duh’)

ii. Type-constructing operators O — these allow specification of unmet requirements:

- a lacking initial argument (precedes lexical head / main word, e.g. subject)
- b lacking final argument (follows lexical head / main word, e.g. object)
- c lacking initial conjunct (precedes conjunction)
- d lacking final conjunct (follows conjunction)
- g lacking gap argument (missing at any position in clause)
- h lacking right-node-raised argument (right-node raising)
- i lacking interrogative pronoun referent
- r lacking relative pronoun referent

iii. Define categories C as follows — clauses with various unmet requirements:

A. every U is in C
B. every C × O × C is in C
C. nothing else is in C

English needs categories for interrog. and relative pronouns, clauses, and gapped clauses:

V-gN gapped finite clause / bare relative (e.g. ‘Kim gave $ε$ to Chris’)
B–gN gapped base-form clause (e.g. ‘Kim give $ε$ to Chris’)
N–iN interrogative pronoun (e.g. ‘what’)
Q–iN content question (e.g. ‘what did Kim give $ε$ to Chris’)
V–iN embedded question (e.g. ‘what Kim gave $ε$ to Chris’)
N–rN relative pronoun (e.g. ‘which’)
V–rN relative clause (e.g. ‘which Kim gave $ε$ to Chris’)

... (and others)

and categories for constituents within clauses:

N–aD common noun, lacking determiner (e.g. ‘risk’)
V–aN3S intransitive verb, lacking subject (e.g. ‘sleeps’)
V–aN3S–bN transitive verb, lacking subject, object (e.g. ‘finds’)
V–aN3S–bN–bN ditransitive verb (e.g. ‘gives’)
V–aN3S–bC bridge verb (e.g. ‘thinks’)
V–aN3S–b(I–aN) raising verb (e.g. ‘seems’)
V–aN3S–b(B–aN) auxiliary verb (e.g. ‘would’)
V–aN3S–bE auxiliary verb (e.g. ‘hopes’)
V–aN3S–b(Q–iN) auxiliary verb (e.g. ‘wonders’)
A–aN adjective / predicative form of verb (e.g. ‘asleep’, ‘knowing’, ‘known’)
A–aN–bN adjectival preposition (e.g. ‘in’, ‘finding’)
A–aN–b (I–aN–gN) tough adjective (e.g. ‘tough’)

4
(b) Observation symbols: \( X = \{ \text{the, a, and, car, ...} \} \)

(c) Start symbols: \( S = \{ S \} \)

(d) Rules in \( R \):

i. lexical items: \( \ldots \) (e.g. \( D \rightarrow \text{the} \))

ii. arguments:
   for \( c \in U \times (\{-a, -b, -c, -d\} \times C)^* \), \( d \in C \):
   \[ c \rightarrow d \quad c-a \quad d \quad c \rightarrow c-b \quad d \]
   e.g.:
   \[ \begin{array}{c}
   \text{N3P} \\
   \text{V-aN3P} \\
   \text{babies} \\
   \text{cry}
   \end{array} \]

iii. modifiers:
   for \( c \in U \times (\{-a, -b, -c, -d\} \times C)^* \), \( u \in U \):
   \[ c \rightarrow c \quad u-a \quad n \quad c \]
   e.g.:
   \[ \begin{array}{c}
   \text{N3P} \\
   \text{N3P} \\
   \text{A-aN} \\
   \text{people} \\
   \text{here}
   \end{array} \]

iv. hypothesize gaps:
   for \( c \in C \), \( d \in C \):
   \[ c-gd \rightarrow c-a \quad d \quad c-gd \rightarrow c-b \quad d \quad c-gd \rightarrow c \]

v. propagate non-local arguments:
   for \( a \rightarrow b \quad c \in R \), \( \psi \in \{-g, -h, -i, -r\} \times C \):
   \[ a\psi \rightarrow b \quad c\psi \quad a\psi \rightarrow b\psi \quad c \quad a\psi \rightarrow b\psi \quad c\psi \]
   e.g.:
   \[ \begin{array}{c}
   \text{something} \\
   \text{V-gN} \\
   \text{N2} \\
   \text{V-aN2-gN} \\
   \text{you} \\
   \text{V-aN2-bN} \\
   \text{ate}
   \end{array} \]

vi. connect gaps to modificands or fillers:
   for \( c \in C \), \( d \in C \), \( e \in C \):
   \[ e \rightarrow e \quad c-e \quad d \quad c-e \quad d \quad e \rightarrow d \quad e \quad c-gd \]
   \[ e \rightarrow e \quad c-gd \quad c \rightarrow d \quad c-gd \]
vii. conjuncts:
for \( c \in C, \ d \in C \):
\[
c \rightarrow d \quad c-d, \quad c-d \rightarrow d \quad c-d, \quad c \rightarrow c-d \quad d
\]
e.g.:

\[ N \rightarrow N \quad N-cN \]
\[ \text{lions} \quad N \quad N-cN \]
\[ \text{tigers} \quad N-cN-dN \quad N \quad \text{and} \quad \text{bears} \]

viii. elided arguments (e.g. determiners):
for \( c \in C, \ d \in C \):
\[
c \rightarrow c-a \quad c \rightarrow c-bd
\]
e.g.:

\[ N \rightarrow N-aD \]
\[ \text{people} \]

ix. sentential forms:
\[ S \rightarrow V \quad \text{declarative sentence} \quad (\text{e.g.} \quad \text{‘Kim slept’}) \]
\[ S \rightarrow B-aN \quad \text{imperative sentence} \quad (\text{e.g.} \quad \text{‘sleep!’}) \]
\[ S \rightarrow Q \quad \text{polar question} \quad (\text{e.g.} \quad \text{‘did Kim sleep?’}) \]
\[ S \rightarrow Q-iN \quad \text{content question} \quad (\text{e.g.} \quad \text{‘what did Kim drive?’}) \]
\[ S \rightarrow N \quad V-gN \quad \text{topicalized sentence} \quad (\text{e.g.} \quad \text{‘that park I like’}) \]
\[ S \rightarrow S \quad V-gS \quad \text{topicalized sentence} \quad (\text{e.g.} \quad \text{‘I ate she said’}) \]
For example:

8. Parsing as deduction:
Can also express rules as inferences in natural deduction:

\[ c \rightarrow d \quad e \quad \iff \quad \frac{d \quad e}{c} \]

Parse trees are then proofs: