Context-Free Grammars

An **alphabet** is a set of symbols —finite or infinite.

Given an alphabet A the set of all *finite* sequences with items in A is written A^*.

A **language** (**over** A) is a subset —finite or infinite— of A^*.

Some languages:

- $\{[0,0],[0,1],[1,0],[1,1]\}$ a finite language over $\{0,1\}$
- $\{0,1\}^* = \{[\,]\,,[0]\,,[1]\,,[0,0]\,,[0,1]\,,[1,0]\,,[1,1]\,,[0,0,0]\,,\cdots\}\,,$ an infinite language over $\{0,1\}$
- $\{[\,]\,,[0]\,,[1]\,,[0,0]\,,[1,1]\,,[0,0,0]\,,[0,1,0]\,,[1,0,1]\,,[1,1,1],\cdots\}$ the infinite set of palindromes over $\{0,1\}$
- $\{[1],[2],[1,\text{‘+’},1],[1,\text{‘+’},2],[2,\text{‘+’},2],[1,\text{‘+’},1,\text{‘+’},1],\cdots\}$ arithmetic expressions using 1, 2, and +

As languages are often infinite sets, we need *finite* descriptions of *infinite* languages.

A **context-free language** (CFL) is a language defined by a *context-free grammar* (below).

Context free languages have numerous applications in data formats, programming languages, communication protocols, etc.

Languages that are too complex to be context free are nevertheless often described by first describing a context free language and then imposing additional restrictions

- E.g. the set of all syntactically correct Java classes is context free

  class C { int i = 13.5 ; }

- The set of all type correct Java classes is *not* context free.

A **context free grammar (CFG)** $(A, N, P, n_{\text{start}})$ consists of

- An alphabet A (i.e. a set of symbols)
- A finite set of **nonterminal symbols** N disjoint from A.
- A finite set of **production rules** of the form $n \to \alpha$, where $n \in N$ and $\alpha \in (N \cup A)^*$
- A **starting nonterminal** $n_{\text{start}} \in N$.

Example

$G_0 = (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (,), -, \sharp\}, \{\text{pn}, \text{d}\}, P_0, \text{pn})$ where P_0 is[1]

$$\{\text{pn} \to (\,\text{d}\,\text{d}\,\text{d}\,)\,\sharp\,\text{d}\,\text{d}\,\text{d} - \text{d}\,\text{d}\,\text{d}\,\text{d},$$
$$\text{d} \to 0, \text{d} \to 1, \text{d} \to 2, \text{d} \to 3, \text{d} \to 4,$$
$$\text{d} \to 5, \text{d} \to 6, \text{d} \to 7, \text{d} \to 8, \text{d} \to 9\}$$

[1] In formal language theory, it is is usual to write sequences just by listing the items. E.g. $\text{d}\,\text{d}\,\text{d}$ instead of $[\text{d}, \text{d}, \text{d}]$. This creates a usually harmless ambiguity between symbols and sequences of length 1. Catenation is written as st rather than $s\hat{}t$. The empty sequence is written as either ϵ or as nothing at all rather than $[\,]$.

For a given grammar $G = (A, N, P, n_{\text{start}})$, define a relation $\Longrightarrow$ on sequences as follows[2]

$\alpha \Longrightarrow \beta$

if and only if

$\alpha, \beta \in (N \cup A)^*$ and

there are sequences $\gamma, \delta, \theta \in (N \cup A)^*$ and an $n \in N$

such that $\alpha = \gamma n \theta$ and $\beta = \gamma \delta \theta$ and $(n \to \delta) \in P$

If $\alpha \Longrightarrow \beta$, we say that we can **derive** β from α in one **step**.

Each **derivation step** $\gamma n \theta \Longrightarrow \gamma \delta \theta$ replaces one occurrence of some nonterminal symbol n with δ where $(n \to \delta) \in P$. For example the following are derivation steps for G_0

$$\begin{aligned} \mathrm{d\,d\,d} &\Longrightarrow \mathrm{8\,d\,d} \\ \mathrm{d\,d\,d} &\Longrightarrow \mathrm{d\,0\,d} \\ \mathrm{pn} &\Longrightarrow (\mathrm{d\,d\,d})\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,d\,d} \end{aligned}$$

Thus each grammar defines a directed graph in which the nodes are elements of $(A \cup N)^*$ and, for each α and β, there is an edge from α to β iff $\alpha \Longrightarrow \beta$.

[2] Using the notation of the rest of the course, we would write the last line of this definition as "such that $\alpha = \gamma\,\hat{}\,[n]\,\hat{}\,\theta$ and $\beta = \gamma\,\hat{}\,\delta\,\hat{}\,\theta$ and $(n \to \delta) \in P$".

A **derivation** is a finite path in this graph. We write $\alpha \overset{*}{\Longrightarrow} \beta$ to mean there is a derivation that starts at α and ends at β. I.e. $\alpha \overset{*}{\Longrightarrow} \beta$ means that we can transform α into β via 0 or more derivation steps.

For example

$$\mathrm{pn} \overset{*}{\Longrightarrow} (\mathrm{d\,d\,9})\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,0\,d}$$

The **language defined by** a CFG $(A, N, P, n_{\mathrm{start}})$ is the set of sequences in $\alpha \in A^*$ such that $n_{\mathrm{start}} \overset{*}{\Longrightarrow} \alpha$.

For example, the language defined by G_0 includes the sequence

$$(7\,0\,9)\sharp 8\,6\,7 - 5\,3\,0\,9$$

To prove this, all we need to do is show 1 derivation (of the many) from pn.

$$\begin{aligned}
\mathrm{pn} &\Longrightarrow (\mathrm{d\,d\,d})\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,d\,d} \Longrightarrow (\mathrm{d\,d\,9})\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,d\,d} \\
&\Longrightarrow (\mathrm{d\,d\,9})\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,0\,d} \Longrightarrow (\mathrm{d\,0\,9})\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,0\,d} \\
&\Longrightarrow (7\,0\,9)\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,0\,d} \Longrightarrow (7\,0\,9)\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,0\,d} \\
&\Longrightarrow (7\,0\,9)\sharp\mathrm{d\,d\,d} - \mathrm{d\,d\,0\,9} \Longrightarrow (7\,0\,9)\sharp\mathrm{d\,d\,d} - \mathrm{d\,3\,0\,9} \\
&\Longrightarrow (7\,0\,9)\sharp 8\,\mathrm{d\,d} - \mathrm{d\,3\,0\,9} \Longrightarrow (7\,0\,9)\sharp 8\,6\,\mathrm{d} - \mathrm{d\,3\,0\,9} \\
&\Longrightarrow (7\,0\,9)\sharp 8\,6\,7 - \mathrm{d\,3\,0\,9} \Longrightarrow (7\,0\,9)\sharp 8\,6\,7 - 5\,3\,0\,9
\end{aligned}$$

Another example. Let $G_1 = (A_1, N_1, P_1, \text{block})$

- $A_1 = \{+, *, /, -, (,), <, :=, \mathbf{if}, \mathbf{then}, \mathbf{while}, \mathbf{do}, \mathbf{else}, \mathbf{end}\} \cup \mathcal{I} \cup \mathcal{N}$, where $\mathcal{I}$ is a finite set of identifiers disjoint from $\{\mathbf{if}, \mathbf{then}, \mathbf{while}, \mathbf{do}, \mathbf{else}, \mathbf{end}\}$ and $\mathcal{N}$ is some finite subset of $\mathbb{N}$.
- $N_1 = \{\text{block}, \text{command}, \text{exp}, \text{comparand}, \text{term}, \text{factor}\}$
- and P_1 contains all of the following production rules[3]

$$
\begin{aligned}
\text{block} &\to \epsilon \\
\text{block} &\to \text{command}\ \text{block} \\
\text{command} &\to i := \text{exp} \qquad \text{for all } i \in \mathcal{I} \\
\text{command} &\to \mathbf{if}\ \text{exp}\ \mathbf{then}\ \text{block}\ \mathbf{else}\ \text{block}\ \mathbf{end\ if} \\
\text{command} &\to \mathbf{while}\ \text{exp}\ \mathbf{do}\ \text{block}\ \mathbf{end\ while} \\
\text{exp} &\to \text{comparand} \\
\text{exp} &\to \text{comparand}\ < \text{comparand} \\
\text{comparand} &\to \text{term} \\
\text{comparand} &\to \text{term} + \text{comparand} \\
\text{comparand} &\to \text{term} - \text{comparand} \\
\text{term} &\to \text{factor} \\
\text{term} &\to \text{factor} * \text{term} \\
\text{term} &\to \text{factor} / \text{term} \\
\text{factor} &\to n \qquad \text{for all } n \in \mathcal{N} \\
\text{factor} &\to i \qquad \text{for all } i \in \mathcal{I} \\
\text{factor} &\to (\,\text{exp}\,)
\end{aligned}
$$

An example of a string in this language is

$$\mathbf{while}\ i < n\ \mathbf{do}\ j := j + i\ i := i + 1\ \mathbf{end\ while}$$

[3] Recall that ϵ means an empty sequence.

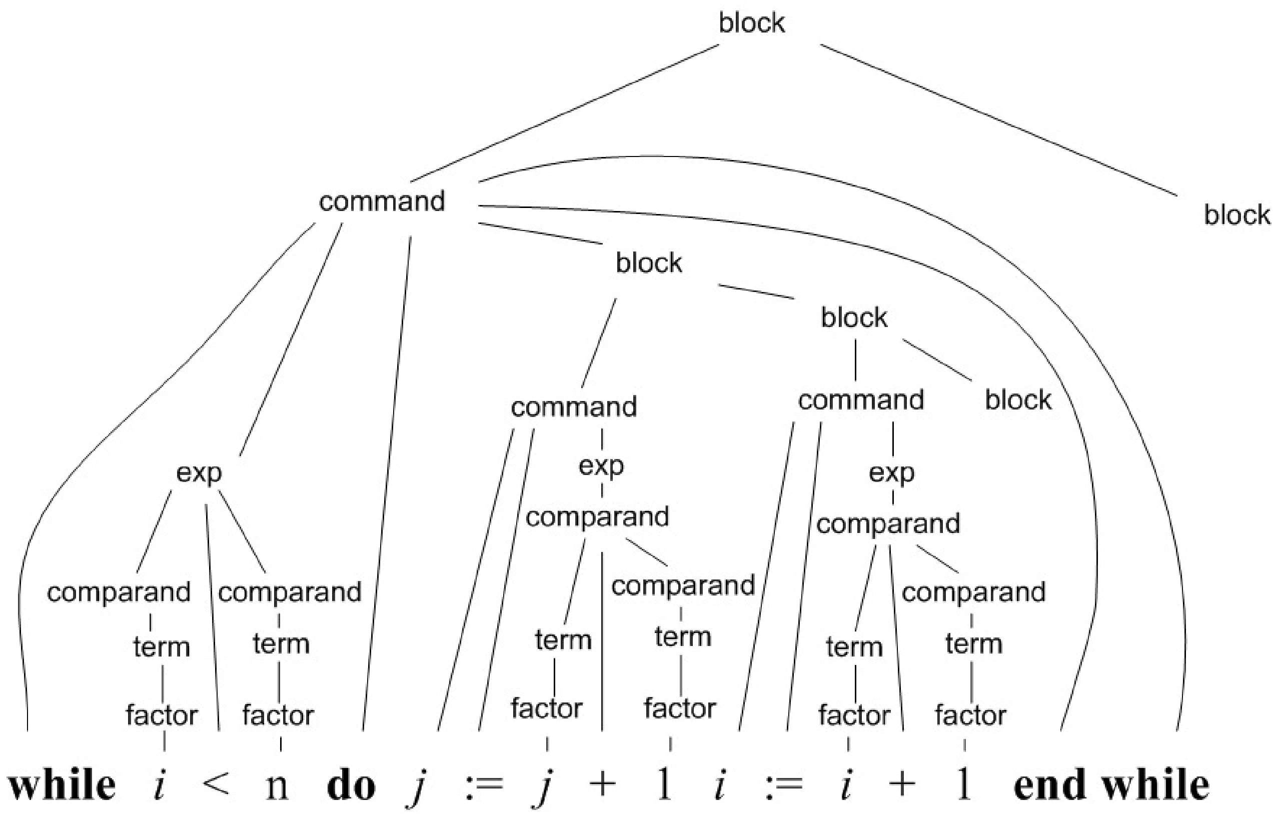
We can show that

$$\textbf{while } i < n \textbf{ do } j := j + i \; i := i + 1 \textbf{ end while}$$

is in the language by showing a derivation.

$$
\begin{array}{ll}
 & \text{block} \\
\Longrightarrow & \text{command block} \\
\Longrightarrow & \textbf{while} \text{ exp } \textbf{do} \text{ block } \textbf{end while} \text{ block} \\
 & \vdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \text{ command block } \textbf{end while} \text{ block} \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := \text{exp block } \textbf{end while} \text{ block} \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := \text{comparand block } \cdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := \text{term} + \text{comparand block } \cdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := \text{factor} + \text{comparand block } \cdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := i + \text{comparand block } \cdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := i + \text{term block } \cdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := i + \text{factor block } \cdots \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := i + 1 \text{ block } \textbf{end while} \text{ block} \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := i + 1 \textbf{ end while} \text{ block} \\
\Longrightarrow & \textbf{while } i < n \textbf{ do } j := j + i \; i := i + 1 \textbf{ end while}
\end{array}
$$

Another way to show that a sequence is in a context free language is with a **parse tree**.



Let's define parse trees.

An ordered tree is a directed tree whose nodes are either leaves (with no children) or branches whose (0 or more) children are arranged in a sequence $children(t)$

A parse tree for a CFG $(A, N, P, n_{\text{start}})$ is a finite ordered tree whose nodes are labelled with symbols from $A \cup N$, such that

- the root is labelled with n_{start} and
- each branch node is labelled with a nonterminal symbol $n \in N$ and has a sequence of children whose labels form a finite sequence α such that $(n \to \alpha) \in P$.

Given a parse tree t the sequence of leaves is given by

```
proc fringe( t )
    if t is a leaf then return [label(t)]
    else // t is a branch
        var α := [ ]
        for u ← children(t) do α := α^fringe(u) end for
        return α
    end if
end fringe
```

Exercise: Show that, for any parse tree t, if $fringe(t) = \alpha$, then $n_{\text{start}} \overset{*}{\Longrightarrow} \alpha$.

Exercise: Show that if $n_{\text{start}} \overset{*}{\Longrightarrow} \alpha$, there is a parse tree t such that $\alpha = fringe(t)$.

Thus, for any $s \in A^*$, s is in the language of G if and only if there is a parse tree t where $s = fringe(t)$.

Exercise: Design a CFG for the language of palindromes in $\{0,1\}^*$.

Exercise: Design a CFG for the language of valid boolean expressions in $\{p, q, r, \wedge, \vee, \Rightarrow, \neg, (,)\}^*$

Exercise: Design a CFG for the language of valid C++ variable declarations in $\{\mathbf{int}, *, [,], (,), ;, a, b, c, 0, 1\}^*$ where a, b, and c are identifiers. E.g.

$$\mathbf{int}\ a(\mathbf{int}, \mathbf{int}*);$$

declares a to be a function, while

$$\mathbf{int}\ (*b[10])();$$

declares b to be an array of 10 pointers to functions returning int results.

Exercise: Look up the definition of regular language.

- Show that every regular language is a context-free language.
- Show that some context free languages are not regular languages.