Recall Theorem 4.52:
Let \( L \) be a context-free language. Then there is a push-down automaton \( M \) with \( L(M) = L \).

The proof of this theorem gives a construction with which one obtains such an automaton given a recursive specification over \( SA \).

**Construction**

Let \( L \) be a context-free language over alphabet \( A \). This means there is a recursive specification over \( SA \) with initial variable \( S \) and \( L(S) = L \).

**Step 1**: Transform this recursive specification into Greibach normal form using variables \( N \).

**Step 2**: Define push-down automaton \( M = (S, A, D, \rightarrow, \uparrow, \downarrow) \) as follows:

1. \( S = \{ \uparrow, \downarrow \} \) (so: two states).
2. \( D = N \) (one stack symbol for each variable).
3. Initial stack: initial variable.
4. For each summand \( a \cdot X \) in the right hand side of a variable \( P \), add a step \( \downarrow P \xrightarrow{a \cdot X} \downarrow \) (for \( a \in A \), \( X \in N^+ \)).
5. For each summand \( 1 \) in the right hand side of variable \( P \), add a step \( \downarrow P \xrightarrow{1} \downarrow \).
6. \( \downarrow \) is a final state.
Intuitively, this transformation stacks the variables of which the right-hand side still needs to be processed.

**Example**

Consider the recursive specification

\[ S = aSc + T \]
\[ T = bTc + 1 \]

(which describes the language \( L = \{a^n b^m c^k \mid k = n + m \} \) from exercise 4.2.10b).

First we transform this specification to Greibach normal form.

- Remove single variable summand \( T \).
  \[ S \rightrightarrows aSc + bTc + 1 \]
  \[ T \rightrightarrows bTc + 1 \]

- Introduce equation \( C = c_1 \).
  \[ S \rightrightarrows aSc + bTcC + 1 \]
  \[ T \rightrightarrows bTcC + 1 \]
  \[ C \rightrightarrows c_1 \]

which is in GNF.

Applying the construction, we get states

1) \( S = \{T, C\} \)
2) \( D = \{S, T, C\} \)
3) \( \epsilon \rightrightarrows S \)
4) \( S, a, SC \)
5) \( S, T, C \)
6) \( \epsilon \) is a final state.

This results in the following automaton:
There also is a construction from push-down automaton to a recursive specification over S.A. This construction is given in Theorem 4. However, the construction of Theorem 4.53 imposes two restrictions on the input automaton:

1. M has exactly one final state, and this state is only entered when the stack content is $\varepsilon$.
2. M has only push and pop transitions, i.e. transitions $S \xrightarrow{a,e} \varepsilon$, $g \in D \cup \{\varepsilon\}$.

This causes a blow-up in the transformation. Therefore we lift these restrictions; Restriction 1 is dropped completely, and we relax restriction 2 to also allow transitions $S \xrightarrow{a,e} g$, $g \in D \cup \{\varepsilon\}$.

Now let $M = (S, \Sigma, D, \rightarrow, \varepsilon, V)$ be such an automaton. The recursive specification is as follows:

1. $N = \{V_{se} | se \in S\} \cup \{V_{sd+} | s, t \in S, de \in D\}$
2. $V_{se}$ has summands $\{a \cdot V_{tdu} | S \xrightarrow{e, a, d} \varepsilon, u \in S\}$
3. $V_{sd+}$ has summands $\{a \cdot V_{t+u} | s \xrightarrow{d, a, e} \varepsilon\}$
4. $V_{tdu}$ has summands $\{a \cdot V_{teu} \cdot V_{vd} | S \xrightarrow{d, a, e} \varepsilon, u \in S\}$
5. $V_{se}$ has summand $1$ if se $\downarrow$
6. $V_{sd+}$ has summands $\{a \cdot V_{tdu} | S \xrightarrow{e, a, e} \varepsilon\}$
7. $V_{se}$ has summands $\{a \cdot V_{+e} | S \xrightarrow{e, a, e} \varepsilon\}$
We apply this construction to the automaton in Figure 4.5.

Recall the automaton:

\[ \begin{align*}
& \overset{a, a, \varepsilon}{\rightarrow} \quad \overset{b, b, \varepsilon}{\rightarrow} \\
& a, a, \varepsilon \quad b, b, \varepsilon \\
& \varepsilon \in D \cup \{c, d\}
\end{align*} \]

We obtain this automaton adheres to our format and we obtain variables:

\[ X = \{ \varepsilon, V_{\varepsilon}, V_{\lambda}, V_{\lambda\varepsilon}, V_{\beta}, V_{\beta\varepsilon}, V_{\lambda\beta}, V_{\lambda\beta\varepsilon} \} \]

With the following equations; note that each summand shows in brackets the rule according to which it is introduced.

\[ V_{\varepsilon} = 1 \tag{5} \]

\[ V_{\varepsilon} = a \cdot V_{\varepsilon} + a \cdot V_{\lambda\varepsilon} + b \cdot V_{\beta\varepsilon} + b \cdot V_{\lambda\beta\varepsilon} \tag{2} \]

\[ V_{\varepsilon} = 1 \tag{5} \]

\[ V_{\varepsilon} = a \cdot V_{\varepsilon} + a \cdot V_{\lambda\varepsilon} \tag{4} \]

\[ + b \cdot V_{\beta\varepsilon} + b \cdot V_{\lambda\beta\varepsilon} \tag{4} \]

\[ + c \cdot V_{\varepsilon} \tag{7} \]

\[ V_{\lambda\varepsilon} = a \cdot V_{\varepsilon} \cdot V_{\lambda} + a \cdot V_{\lambda\varepsilon} \cdot V_{\lambda} \tag{4} \]

\[ + b \cdot V_{\beta\varepsilon} \cdot V_{\beta\varepsilon} + b \cdot V_{\lambda\beta\varepsilon} \cdot V_{\lambda\beta\varepsilon} \tag{4} \]

\[ + c \cdot V_{\lambda\varepsilon} \tag{7} \]

\[ V_{\beta\varepsilon} = a \cdot V_{\varepsilon} \cdot V_{\beta} + a \cdot V_{\beta\varepsilon} \cdot V_{\beta} \tag{4} \]

\[ + b \cdot V_{\varepsilon} \cdot V_{\varepsilon} + b \cdot V_{\beta\varepsilon} \cdot V_{\beta\varepsilon} \tag{4} \]

\[ + c \cdot V_{\beta\varepsilon} \tag{7} \]

\[ V_{\lambda\beta\varepsilon} = a \cdot V_{\varepsilon} \cdot V_{\lambda\beta} + a \cdot V_{\lambda\varepsilon} \cdot V_{\lambda\beta} \tag{4} \]

\[ + b \cdot V_{\beta\varepsilon} \cdot V_{\beta\varepsilon} + b \cdot V_{\lambda\beta\varepsilon} \cdot V_{\lambda\beta\varepsilon} \tag{4} \]

\[ + c \cdot V_{\lambda\beta\varepsilon} \tag{7} \]

\[ V_{\lambda\beta\varepsilon} = \text{analogous to } V_{\lambda\varepsilon} \]
\[ V_{\text{val}} = a \cdot 1 \quad (3) \]
\[ V_{\text{vbl}} = b \cdot 1 \quad (3) \]
\[ V_{\text{van}} = 0 \]
\[ V_{\text{vbr}} = 0 \]

Observe that \( V_{\text{van}} \) and \( V_{\text{vbr}} \) are non-productive and can be removed. As a result, we can also remove 0's in the other equations, resulting in the following specification.

\[ V_{\text{vE}} = a \cdot V_{\text{van}} + a \cdot V_{\text{val}} + b \cdot V_{\text{vbl}} + b \cdot V_{\text{vbr}} + T \cdot V_{\text{vbl}} \]
\[ V_{\text{vE}} = 1 \]
\[ V_{\text{vap}} = a \cdot V_{\text{van}} \cdot V_{\text{van}} + b \cdot V_{\text{vbr}} \cdot V_{\text{van}} \]
\[ V_{\text{vbp}} = a \cdot V_{\text{van}} \cdot V_{\text{vbr}} + b \cdot V_{\text{vbr}} \cdot V_{\text{vbp}} \]
\[ V_{\text{val}} = a \cdot V_{\text{van}} \cdot V_{\text{val}} + a \cdot V_{\text{val}} \cdot V_{\text{val}} + b \cdot V_{\text{vbr}} \cdot V_{\text{val}} + b \cdot V_{\text{val}} \cdot V_{\text{val}} + T \cdot V_{\text{vval}} \]
\[ V_{\text{vbl}} = a \cdot V_{\text{van}} \cdot V_{\text{vbl}} + a \cdot V_{\text{val}} \cdot V_{\text{vbl}} + b \cdot V_{\text{vbr}} \cdot V_{\text{vbl}} + b \cdot V_{\text{vbl}} \cdot V_{\text{vbl}} + T \cdot V_{\text{vbl}} \]
\[ V_{\text{vbl}} = a \cdot 1 \]
\[ V_{\text{vbl}} = b \cdot 1 \]
Now, using the observation that $V_{\uparrow \uparrow}$ and $V_{\downarrow \downarrow}$ are non-productive, we can further simplify to:

$V_{\downarrow \uparrow} = a \cdot V_{\uparrow \downarrow} + b \cdot V_{\downarrow \downarrow} + c \cdot V_{\downarrow \downarrow}$

$V_{\downarrow \downarrow} = 1$

$V_{\uparrow \downarrow} = a \cdot V_{\uparrow \downarrow} \cdot V_{\downarrow \downarrow} + b \cdot V_{\downarrow \downarrow} \cdot V_{\downarrow \downarrow} + c \cdot V_{\downarrow \downarrow}$

$V_{\downarrow \downarrow} = a \cdot V_{\uparrow \downarrow} \cdot V_{\downarrow \downarrow} + b \cdot V_{\downarrow \downarrow} \cdot V_{\downarrow \downarrow} + c \cdot V_{\downarrow \downarrow}$

$V_{\downarrow \downarrow} = a \cdot 1

V_{\downarrow \downarrow} = b \cdot 1$

The intuition behind this translation is as follows. We start in $V_{\downarrow \uparrow}$, encoding the initial state with an empty stack (which is indeed the initial state!). Variable $V_{\downarrow \downarrow}$ encodes that from state $s$, there is a desire to reach state $t$ with such that symbol $d$ is on the top of the stack if you reach $t$. In push transitions, this is done by going through an intermediate state; in pop transitions, you have reached some desired state, and you remove a recursion variable. In effect, the stack is encoded by a sequence of recursion variables; the number of stack variables to be processed changes only in push and pop transitions.