Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Talen en Compilers

2023 - 2024

David van Balen

Department of Information and Computing Sciences
Utrecht University

2024-01-16

12. Pumping lemma's and context-free languages

This lecture

wooclap questions

Pumping lemma's and context-free languages

Pumping lemma for regular languages

12.1 Pumping lemma for regular languages

How to prove that a language is not regular?

Generally, proving that a language does not belong to a certain class is much more difficult than proving that it does.

In the case of regular languages,

- **to show that a language is regular**, we have to give one regular grammar (or regular expression, or DFA, or NFA) that describes the language;

How to prove that a language is not regular?

Generally, proving that a language does not belong to a certain class is much more difficult than proving that it does.

In the case of regular languages,

- **to show that a language is regular**, we have to give one regular grammar (or regular expression, or DFA, or NFA) that describes the language;
- **to show that a language is not regular**, we have to prove that no regular grammar (or regular expression, or DFA, or NFA) is possible that describes the language.

The strategy

We proceed in the following steps:

1. we expose a **limitation** in the formalism (in this case, in the concept of finite state automata);
2. from this limitation, we derive a **property** that all languages in the class (in this case, regular languages) must have;
3. therefore, if a language does not have that property, it cannot be in the class.

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

How many different states do we visit...

- if the string has length 0?

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

How many different states do we visit...

- if the string has length 0?
  – One (the start state).

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

How many different states do we visit...

- if the string has length 0?
  – One (the start state).
- if the string has length 1?

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

How many different states do we visit...

- if the string has length 0?
  – One (the start state).
- if the string has length 1?
  – Two or one. One if for the given terminal, the start state has a transition to itself, i.e., we walk through a loop.

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

How many different states do we visit...

- if the string has length 0?
  – One (the start state).
- if the string has length 1?
  – Two or one. One if for the given terminal, the start state has a transition to itself, i.e., we walk through a loop.
- if the string has length 2?

Loops in deterministic finite state automata

Assume we have a deterministic finite state automaton, and we read a string that is accepted.

How many different states do we visit...

- if the string has length 0?
  – One (the start state).
- if the string has length 1?
  – Two or one. One if for the given terminal, the start state has a transition to itself, i.e., we walk through a loop.
- if the string has length 2?
  – Three or two or one. If less than three, we visit at least one state twice, i.e., walk through a loop.

Finite state automata are finite

Any finite state automaton has a finite number of states.
Assume we have one with n states.

Question

How many different states do we visit while reading a string that is accepted and has length n?

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Finite state automata are finite

Any finite state automaton has a finite number of states.
Assume we have one with n states.

Question

How many different states do we visit while reading a string that is accepted and has length n?

Answer

According to the previous considerations, $n + 1$ or less, and if less, we traverse a loop.
But there are only n states, so we cannot traverse $n + 1$ different states. Therefore, we **must** traverse a loop.

The strategy – revisited

We proceed in the following steps:

1. we expose a limitation in the formalism (in this case, in the concept of finite state automata);
2. from this limitation, we derive a property that all languages in the class (in this case, regular languages) must have;
3. therefore, if a language does not have that property, it cannot be in the class.

The strategy – revisited

We proceed in the following steps:

1. we expose a limitation in the formalism (in this case, in the concept of finite state automata);
2. from this limitation, we derive a property that all languages in the class (in this case, regular languages) must have;
3. therefore, if a language does not have that property, it cannot be in the class.

We have done the first step. We have found a limitation in the formalism. Now we have to derive a property for all regular languages from that.

A property of loops

Question

What can we say about a loop in a finite state automaton?

A property of loops

Question

What can we say about a loop in a finite state automaton?

Answer

We can traverse it arbitrarily often.

A property of loops

Question

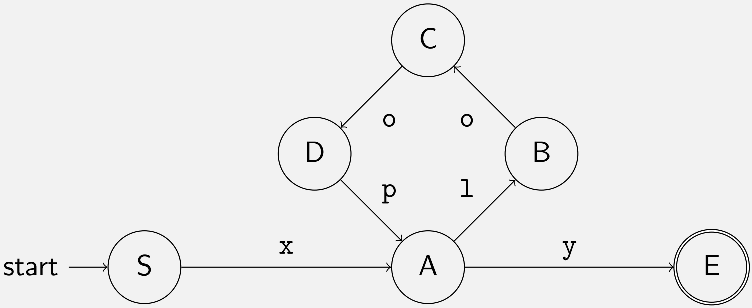
What can we say about a loop in a finite state automaton?

Answer

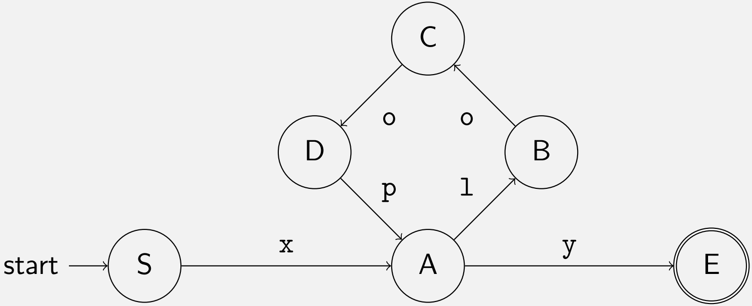
We can traverse it arbitrarily often.

To be more precise: if we have a word that is accepted and traverses the loop once, then the words that follow the same path and traverse the loop any other number of times are also accepted.

Example



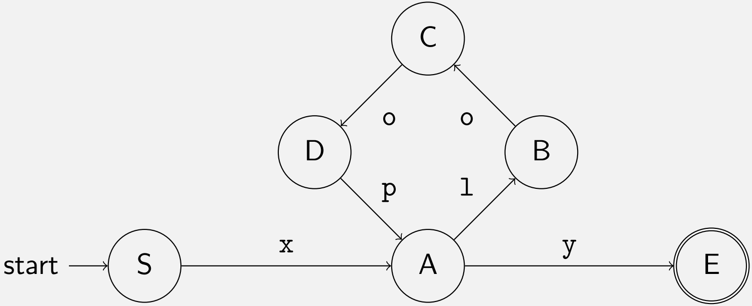
Example



The automaton accepts:

`xloopy` (1 loop traversal)

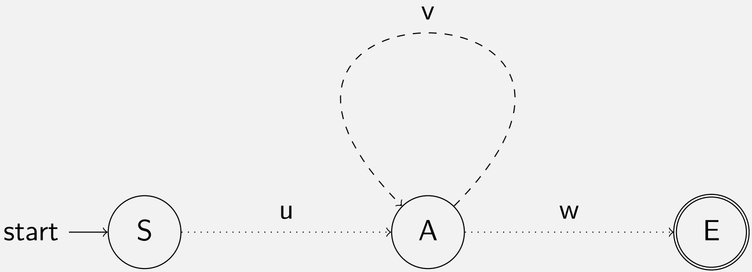
Example



The automaton accepts:

`xy`	(0 loop traversals)
`xloopy`	(1 loop traversal)
`xlooploopy`	(2 loop traversals)
`xlooplooploopy`	(3 loop traversals)
...	

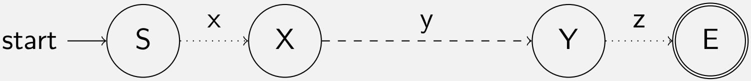
The general situation



- This is an excerpt of the automaton. There may be other nodes and edges.
- Both u and w may be empty (i.e. A and S or A and E may be the same state), but v is not empty – there is a proper loop.
- All words of the form uv^iw for $i \in \mathbb{N}$ are accepted.

Generalizing even more

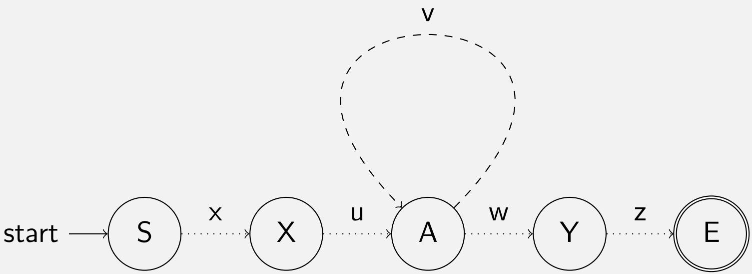
A loop has to occur in every **subword** of at least length n:



- Assume we have an accepted word xyz where subword y is of at least length n.

Generalizing even more

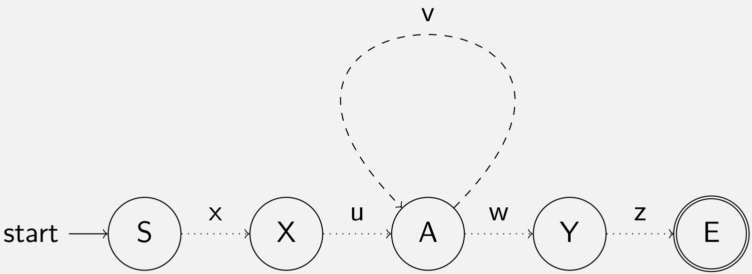
A loop has to occur in every **subword** of at least length n:



- Assume we have an accepted word xyz where subword y is of at least length n.
- Then y has to be of form uvw where v is not empty and corresponds to a loop.

Generalizing even more

A loop has to occur in every **subword** of at least length n:



- Assume we have an accepted word xyz where subword y is of at least length n.
- Then y has to be of form uvw where v is not empty and corresponds to a loop.
- All words of the form xuv^iwz for $i \in \mathbb{N}$ are accepted.

A property of all regular languages

Pumping Lemma for regular languages

For every regular language L,

A property of all regular languages

Pumping Lemma for regular languages

For every regular language L,
there exists an $n \in \mathbb{N}$

- (corresponding to the number of states in the automaton)

A property of all regular languages

Pumping Lemma for regular languages

For every regular language L,
there exists an $n \in \mathbb{N}$

- (corresponding to the number of states in the automaton)

such that for every word xyz in L with $|y| \geqslant n$,

- (this holds for every long substring of every word in L)

A property of all regular languages

Pumping Lemma for regular languages

For every regular language L,
there exists an $n \in \mathbb{N}$

- (corresponding to the number of states in the automaton)

such that for every word xyz in L with $|y| \geqslant n$,

- (this holds for every long substring of every word in L)

we can split y into three parts, $y = uvw$, with $|v| > 0$,

- (v is a loop)

A property of all regular languages

Pumping Lemma for regular languages

For every regular language L,
there exists an $n \in \mathbb{N}$

such that for every word xyz in L with $|y| \geqslant n$,

we can split y into three parts, $y = uvw$, with $|v| > 0$,

such that for every $i \in \mathbb{N}$, we have $xuv^iwz \in L$.

The strategy – revisited

We proceed in the following steps:

1. we expose a limitation in the formalism (in this case, in the concept of finite state automata);
2. from this limitation, we derive a property that all languages in the class (in this case, regular languages) must have;
3. therefore, if a language does not have that property, it cannot be in the class.

The strategy – revisited

We proceed in the following steps:

1. we expose a limitation in the formalism (in this case, in the concept of finite state automata);
2. from this limitation, we derive a property that all languages in the class (in this case, regular languages) must have;
3. therefore, if a language does not have that property, it cannot be in the class.

We have done the first two steps. We have found a limitation in the formalism, and derived a property that all regular languages must have.

Using the pumping lemma

In order to show that a language is not regular, we show that it does not have the pumping lemma property as follows:

- We assume that the language is regular.
- We use the pumping lemma to derive a word that must be in the language, but is not:
  - find a word xyz in L with $|y| \geqslant n$,
  - from the pumping lemma there must be a loop in y,
  - but repeating this loop, or omitting it, takes us outside of the language.
- The contradiction means that the language cannot be regular.

Using the pumping lemma – strategy

- For **every** natural number n,
  - because you don't know what the value of n is
- find a word xyz in L with $|y| \geqslant n$ (**you choose** the word),
- such that for **every** splitting $y = uvw$ with $|v| > 0$,
  - because you don't know where the loop may be
- there exists a number i (**you figure out** the number),
- such that $xuv^iwz \notin L$ (you have to **prove** it).

Wooclap questions

Exercise

For each of these languages:

- if it is regular, give an automaton or regular expr. for it;
- if not, use the pumping lemma to prove it.

1. $L = \{a^m b^n \mid m, n \in \mathbb{N}\}$
2. $L = \{a^m b^n \mid m, n \in \mathbb{N}, m < n\}$
3. $L = \{a^m b^n \mid m, n < 1000, m < n\}$
4. $L = \{a^m b^n \mid m < 1000, m < n\}$

Context-free grammars

A context-**free** grammar consists of a sequence of productions:

> N → x

- the **left hand side** is always a **nonterminal**,
- the right hand side is any sequence of terminals and nonterminals.

One nonterminal of the grammar is the start symbol.

Context-sensitive grammars

Context-**sensitive** grammars drop the restriction on the left hand side:

> a N b $\rightarrow$ x

Context-sensitive grammars

Context-**sensitive** grammars drop the restriction on the left hand side:

> a N b $\rightarrow$ x

Context-sensitive grammars are as **powerful** as any other computing formalism:

- Turing machines,
- λ-calculus.

Not interesting from a parsing perspective.

The strategy – revisited

If we want to prove that a certain language is **not context-free**, we can apply the same strategy as for regular languages:

- we expose a limitation in the formalism (in this case, in the concept of context-free grammars);
- from this limitation, we derive a property that all languages in the class (in this case, context-free languages) must have;
- therefore, if a language does not have that property, it cannot be in the class.

The strategy – revisited

If we want to prove that a certain language is **not context-free**, we can apply the same strategy as for regular languages:

- we expose a limitation in the formalism (in this case, in the concept of context-free grammars);
- from this limitation, we derive a property that all languages in the class (in this case, context-free languages) must have;
- therefore, if a language does not have that property, it cannot be in the class.

This time, we analyze parse trees rather than finite state automata.

Grammars and parse trees

For every word in the language, there is a parse tree.

We observe:

Grammars and parse trees

For every word in the language, there is a parse tree.

We observe:

- We can produce parse trees of arbitrary depth if we find words in the language that are long enough, because the number of children per node is bounded by the maximum length of a right hand side of a production.

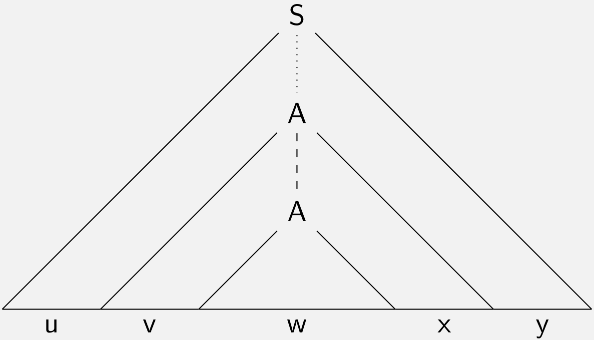
Grammars and parse trees

For every word in the language, there is a parse tree.

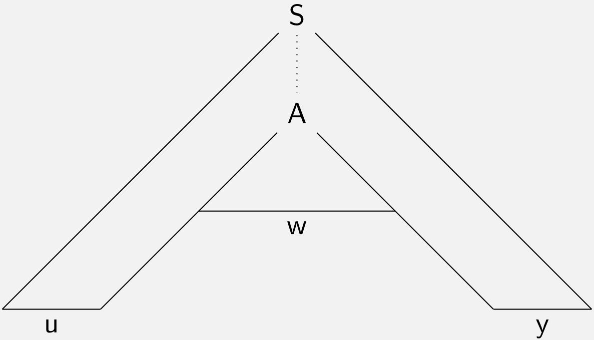
We observe:

- We can produce parse trees of arbitrary depth if we find words in the language that are long enough, because the number of children per node is bounded by the maximum length of a right hand side of a production.
- Once a path from a leaf to the root has more than n internal nodes, where n is the number of nonterminals in the grammar, one nonterminal has to occur twice on such a path.

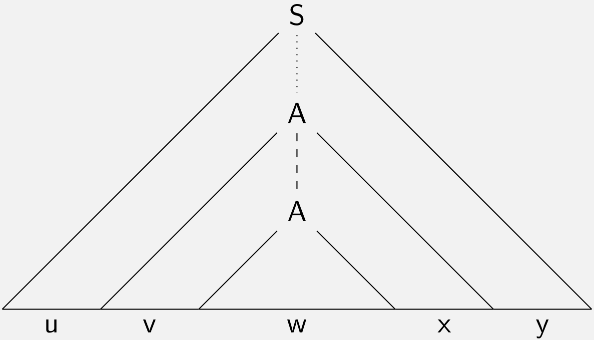
The situation



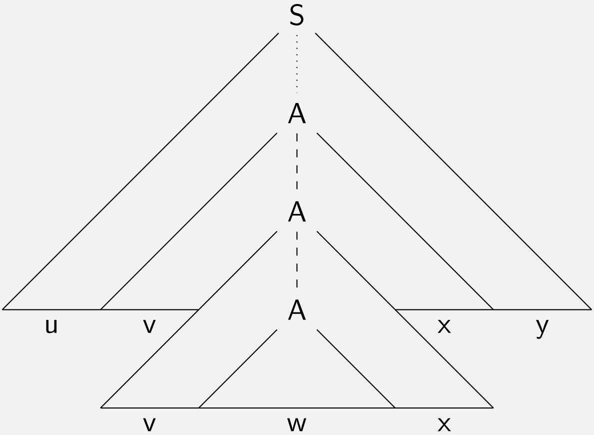
The situation



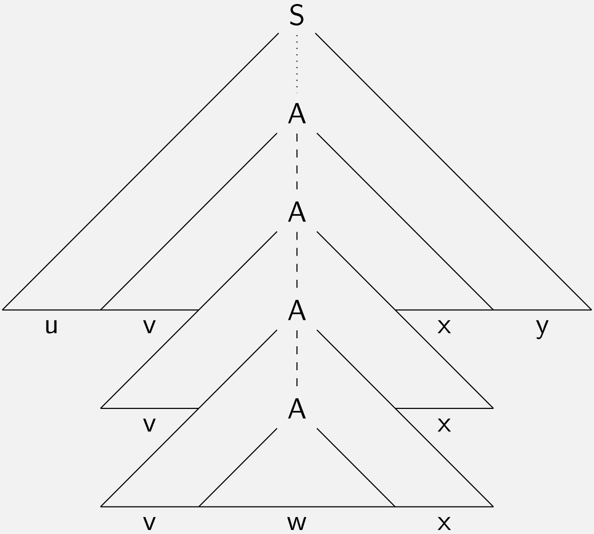
The situation



The situation



The situation



The situation – contd.

If the word is long enough, we have a derivation of the form

> $S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy$

where $|vx| > 0$.

The situation – contd.

If the word is long enough, we have a derivation of the form

> $S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy$

where $|vx| > 0$.

Because the grammar is context-free, this implies that

> $A \Rightarrow^* vAx$
> $A \Rightarrow^* w$

The situation – contd.

If the word is long enough, we have a derivation of the form

> $S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy$

where $|vx| > 0$.

Because the grammar is context-free, this implies that

> $A \Rightarrow^* vAx$
> $A \Rightarrow^* w$

We can thus derive

> $S \Rightarrow^* uAy \Rightarrow^* uv^iwx^iy$

for any $i \in \mathbb{N}$.

The lemma

Pumping lemma for context-free languages

For every context-free language L,

The lemma

Pumping lemma for context-free languages

For every context-free language L,

- there exists a number $n \in \mathbb{N}$ such that

The lemma

Pumping lemma for context-free languages

For every context-free language L,

- there exists a number $n \in \mathbb{N}$ such that
- for every word $z \in L$ with $|z| \geqslant n$,

The lemma

Pumping lemma for context-free languages

For every context-free language L,

- there exists a number $n \in \mathbb{N}$ such that
- for every word $z \in L$ with $|z| \geqslant n$,
- we can split z into five parts, $z = uvwxy$, with $|vx| > 0$ and $|vwx| \leqslant n$, such that

The lemma

Pumping lemma for context-free languages

For every context-free language L,

- there exists a number $n \in \mathbb{N}$ such that
- for every word $z \in L$ with $|z| \geqslant n$,
- we can split z into five parts, $z = uvwxy$, with $|vx| > 0$ and $|vwx| \leqslant n$, such that
- for every $i \in \mathbb{N}$, we have $uv^iwx^iy \in L$.

The lemma

Pumping lemma for context-free languages

For every context-free language L,

- there exists a number $n \in \mathbb{N}$ such that
- for every word $z \in L$ with $|z| \geqslant n$,
- we can split z into five parts, $z = uvwxy$, with $|vx| > 0$ and $|vwx| \leqslant n$, such that
- for every $i \in \mathbb{N}$, we have $uv^iwx^iy \in L$.

The n lets us limit the size of the part that gets pumped, similar to how the pumping lemma for regular languages lets us choose the subword that contains the loop.

Using the pumping lemma

- For **every** of number n,
- find a word z in L with $|z| \geqslant n$ (**you choose** the word),
- such that for **every** splitting $z = uvwxy$ with $|vx| > 0$ and $|vwx| \leqslant n$,
- there exists a number i (**you choose** the number),
- such that $uv^iwx^iy \notin L$ (you have to **prove** it).

An example

Theorem

The language $L = \{a^m b^m c^m \mid m \in \mathbb{N}\}$ is not context-free.

An example

Theorem

The language $L = \{a^m b^m c^m \mid m \in \mathbb{N}\}$ is not context-free.

Let n be any number.

An example

Theorem

The language $L = \{a^m b^m c^m \mid m \in \mathbb{N}\}$ is not context-free.

Let n be any number.

We then consider the word $z = a^n b^n c^n$.

An example

Theorem

The language $L = \{a^m b^m c^m \mid m \in \mathbb{N}\}$ is not context-free.

Let n be any number.

We then consider the word $z = a^n b^n c^n$.

From the pumping lemma, we learn that we can pump z, and that the part that gets pumped is smaller than n.

An example

Theorem
The language $L = \{a^m b^m c^m \mid m \in \mathbb{N}\}$ is not context-free.

Let n be any number.

We then consider the word $z = a^n b^n c^n$.

From the pumping lemma, we learn that we can pump z, and that the part that gets pumped is smaller than n.

The part being pumped can thus not contain a's, b's **and** c's at the same time, and is not empty either. In all these cases, we pump out of the language (for any $i \neq 1$).

Wooclap questions!

For more practice exercises, see the lecture notes

Normal forms

Context-free grammars can be wildly complex, in general.

But all of them can be brought into more normalised forms.

- We call them **normal forms**.

We get to them by applying **grammar transformations** (see lecture 4).

Chomsky Normal Form

A context-free grammar is in **Chomsky Normal Form** if each production rule has one of these forms:

$$\mathsf{A} \rightarrow \mathsf{B}\ \mathsf{C}$$
$$\mathsf{A} \rightarrow \mathtt{x}$$
$$\mathsf{S} \rightarrow \varepsilon$$

where A, B, and C are nonterminals, `x` is a terminal, and S is the start symbol of the grammar. Also, B and C cannot be S.

Chomsky Normal Form

A context-free grammar is in **Chomsky Normal Form** if each production rule has one of these forms:

> A $\rightarrow$ B C
> A $\rightarrow$ $\mathtt{x}$
> S $\rightarrow$ ε

where A, B, and C are nonterminals, $\mathtt{x}$ is a terminal, and S is the start symbol of the grammar. Also, B and C cannot be S.

- No rule produces ε except (possibly) from the start.
- No chain rules of the form A $\rightarrow$ B.
- Parse trees are always binary.

Greibach Normal Form

A context-free grammar is in **Greibach Normal Form** if each production rule has one of these forms:

> $\mathsf{A} \rightarrow \mathtt{x}\mathsf{A}_1\mathsf{A}_2 \ldots A_n$
> $\mathsf{S} \rightarrow \varepsilon$

where A, A_1, ..., A_n are nonterminals ($n \geqslant 0$), $\mathtt{x}$ is a terminal, and S is the start symbol of the grammar and does not occur in any right hand side.

Greibach Normal Form

A context-free grammar is in **Greibach Normal Form** if each production rule has one of these forms:

> $\mathsf{A} \rightarrow \mathrm{x}\mathsf{A}_1\mathsf{A}_2 \ldots A_n$
> $\mathsf{S} \rightarrow \varepsilon$

where A, A_1, ..., A_n are nonterminals ($n \geqslant 0$), $\mathtt{x}$ is a terminal, and S is the start symbol of the grammar and does not occur in any right hand side.

- At most one rule produces ε, and only from the start.
- No left recursion.
- A derivation of a word of length n has exactly n rule applications (except ε).
- Generalizes GNF for regular grammars (where $n \leqslant 1$)