有限オートマトンで認識できない言語が存在する

⇓

より強力な計算モデルが必要

⇓

- **プッシュダウンオートマトン**
- **チューリングマシン**

定理：

L：**正規言語**

⇕

L **が或る有限オートマトンで認識される**

定理：

L：**文脈自由言語**

⇕

L **が或るプッシュダウンオートマトンで認識される**

本質的な違いは？

文脈自由言語は再帰 (recursion) を記述できる

定理：

L：正規言語

$\Updownarrow$

L が或る有限オートマトンで認識される

定理：

L：文脈自由言語

$\Updownarrow$

L が或るプッシュダウンオートマトンで認識される

本質的な違いは？

文脈自由言語は再帰 (recursion) を記述できる

文脈自由言語と再帰

- $S \rightarrow aSb|\varepsilon$

```
S(){
  either
    "";
  or
    { "a"; S() ; "b"; }
}

main(){
  S();
}
```

再帰：関数 S() の中で、自分自身を呼び出す

計算機での関数呼出・再帰の実現

関数呼出は原理的には次の仕組みで行なっている

- **現在の実行番地（戻る場所）を覚えておく**
- **関数を実行する**
- **関数を実行し終えたら、覚えていた実行番地に戻って呼出側の実行再開**

再帰呼出では呼出す度に覚えておく番地が増える

$\longrightarrow$ スタックに積んで覚えておく（関数呼出の際に番地を push、戻ったら pop）

計算機での関数呼出・再帰の実現

関数呼出は原理的には次の仕組みで行なっている

- 現在の実行番地（戻る場所）を覚えておく
- 関数を実行する
- 関数を実行し終えたら、
  覚えていた実行番地に戻って呼出側の実行再開

再帰呼出では呼出す度に覚えておく番地が増える

$\longrightarrow$ スタックに積んで覚えておく
（関数呼出の際に番地を push、戻ったら pop）

計算機での関数呼出・再帰の実現

関数呼出は原理的には次の仕組みで行なっている

- **現在の実行番地（戻る場所）を覚えておく**
- **関数を実行する**
- **関数を実行し終えたら、**
  覚えていた実行番地に戻って呼出側の実行再開

再帰呼出では呼出す度に覚えておく番地が増える

$\longrightarrow$ **スタックに積んで覚えておく**
（関数呼出の際に番地を push、戻ったら pop）

正規言語における再帰

正規表現：$(aa)^*$

- $S \rightarrow aaS|\varepsilon$

```
S(){
  either
    "";
  or
    { "aa"; S() ; }
}

main(){
  S();
}
```

$\longrightarrow$ 末尾再帰の除去

```
main(){
  loop {
    "aa";
  }
}
```

繰返しで記述可能
（再帰は不要）

正規言語における再帰

正規表現： $(aa)^*$

- $S \rightarrow aaS|\varepsilon$

```
S(){
  either
    "";
  or
    { "aa"; S() ; }
}

main(){
  S();
}
```

$\longrightarrow$ **末尾再帰の除去**

```
main(){
  loop {
    "aa";
  }
}
```

繰返しで記述可能
（再帰は不要）

正規言語・文脈自由言語と再帰

- 正規言語は繰返しを記述できる
- 文脈自由言語は再帰を記述できる
- 再帰の実装にはスタックを要す
- 文脈自由言語の生成規則は次の形に出来る
  - $\star\ X \longrightarrow YZ\ (X, Y, Z \in V)$
  - $\star\ X \longrightarrow x\ (X \in V, x \in \Sigma_\varepsilon)$

  (Chomsky の標準形)
- 正規言語の生成規則は次の形に出来る
  - $\star\ X \longrightarrow xY\ (X, Y \in V, x \in \Sigma)$
  - $\star\ X \longrightarrow x\ (X \in V, x \in \Sigma_\varepsilon)$

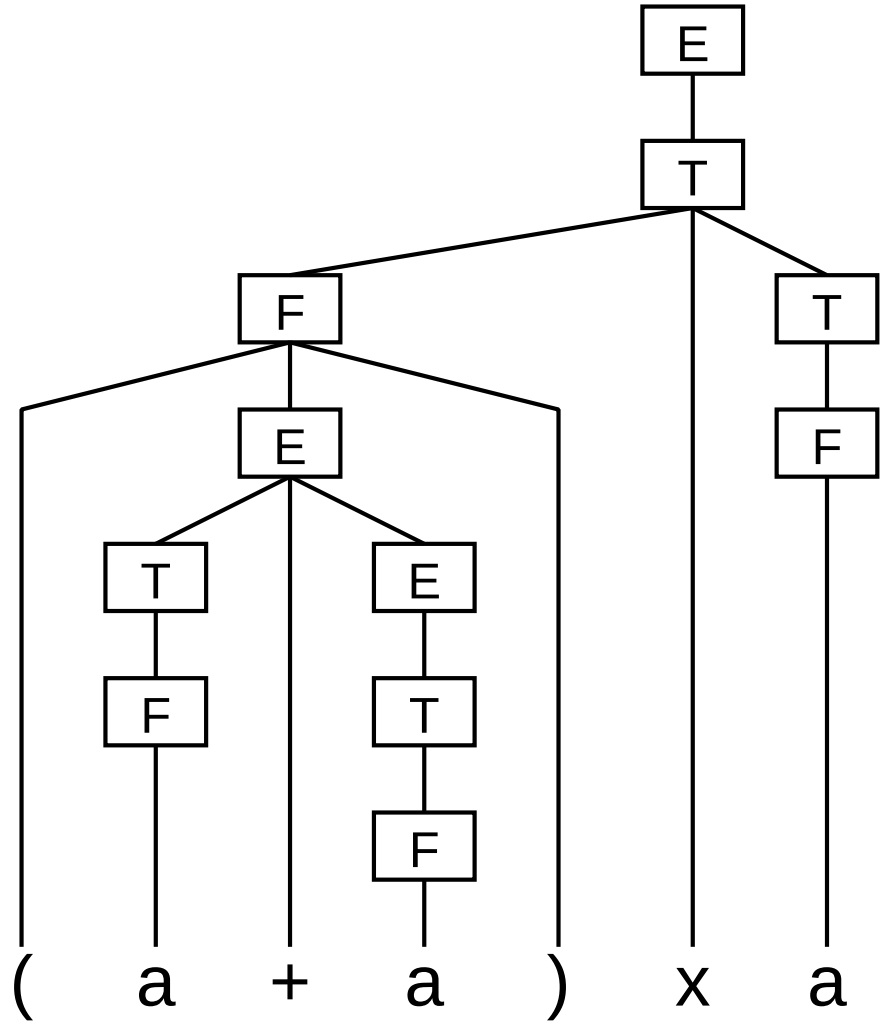
  特に、末尾再帰であり再帰の除去可能

構文解析木

生成規則の適用過程を木で表したもの

$G = (V, \Sigma, R, E)$

- $V = \{E, T, F\}$
- $\Sigma = \{a, +, \times, (,)\}$
- R:
  - $E \longrightarrow T \mid T + E$
  - $T \longrightarrow F \mid F \times T$
  - $F \longrightarrow a \mid (E)$



文脈自由言語の Pumping Lemma

文脈自由言語 A に対し、

$\exists n \in \mathbb{N}$:

$\forall w \in A, |w| \geq n$:

$\exists u, v, x, y, z \in \Sigma^* : w = uvxyz$

(1) $vy \neq \varepsilon$ **（即ち** $v \neq \varepsilon$ **または** $y \neq \varepsilon$**）**

(2) $|vxy| \leq n$

(3) $\forall k \geq 0 : uv^k xy^k z \in A$

文脈自由言語の例

回文全体の成す言語は文脈自由

- $S \to aSa|bSb|a|b|\varepsilon$

問：回文全体の成す言語を認識する
プッシュダウンオートマトンを構成せよ

文脈自由言語の例

回文全体の成す言語は文脈自由

- $S \to aSa|bSb|a|b|\varepsilon$

問：回文全体の成す言語を認識する
プッシュダウンオートマトンを構成せよ

文脈自由言語の例

回文全体の成す言語は文脈自由

- $S \to aSa|bSb|a|b|\varepsilon$

問：回文全体の成す言語を認識する
プッシュダウンオートマトンを構成せよ

プッシュダウンオートマトンでは認識できない言語の例

同じ文字列 2 回の繰返しから成る文字列全体

$$A = \{ww | w \in \Sigma^*\}$$

入力を読み直せないのが弱点

$\longrightarrow$ より強力な計算モデルが必要

プッシュダウンオートマトンでは認識できない言語の例

同じ文字列 2 回の繰返しから成る文字列全体

$$A = \{ww | w \in \Sigma^*\}$$

入力を読み直せないのが弱点

$\longrightarrow$ **より強力な計算モデルが必要**

プッシュダウンオートマトンでは認識できない言語の例

同じ文字列 2 回の繰返しから成る文字列全体

$$A = \{ww | w \in \Sigma^*\}$$

入力を読み直せないのが弱点

$\longrightarrow$ **より強力な計算モデルが必要**

一つの方法としては、

入力を覚えておくために
プッシュダウンスタックをもう一つ
使えることにする

実際これで真により強い計算モデルが得られる

しかし、通常はこれと同等な
次のような計算モデルを考える
… チューリングマシン

一つの方法としては、

入力を覚えておくために
プッシュダウンスタックをもう一つ
使えることにする

実際これで真により強い計算モデルが得られる

しかし、通常はこれと同等な
次のような計算モデルを考える
... チューリングマシン

チューリングマシン

- 有限個の内部状態を持つ
- 入力データはテープ上に一区画一文字づつ書き込まれて与えられる
- データを読み書きするヘッドがテープ上を動く
- 遷移関数は次の形：
  内部状態とヘッドが今いる場所の文字とによって、その場所の文字を書き換え、次の内部状態に移り、ヘッドを左か右かに動かす
- 受理状態または拒否状態に達したら停止するが、停止しないこともある