Regular Expressions

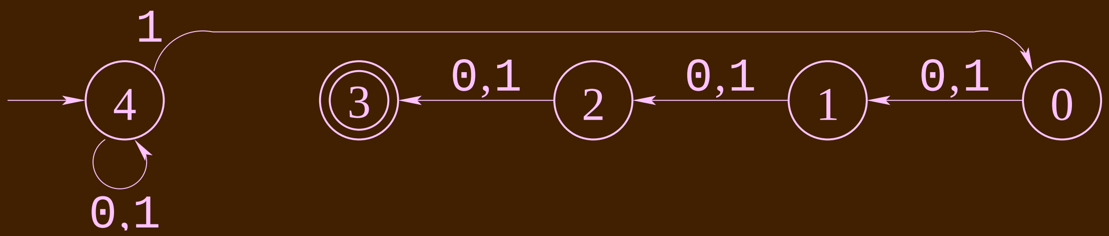
Mark Greenstreet, CpSc 421, Term 1, 2006/07

Lecture Outline

Regular Expressions

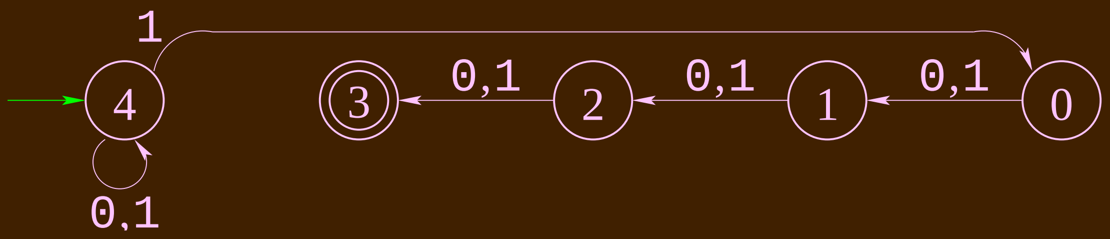
- Finishing the Equivalence of NFAs and DFAs
- Introducing Regular Expresssions

Another NFA Example



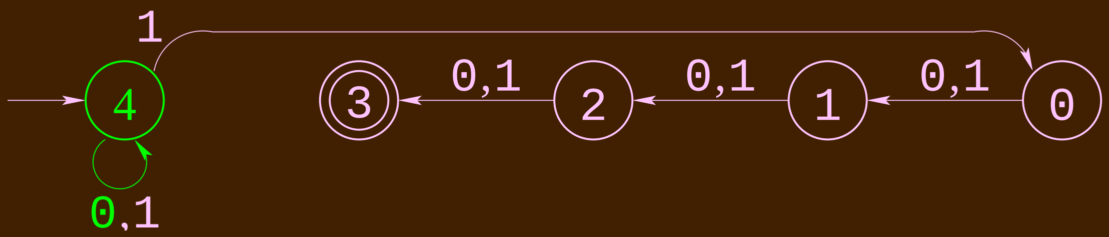
- Processing the string: `0101101`

Another NFA Example



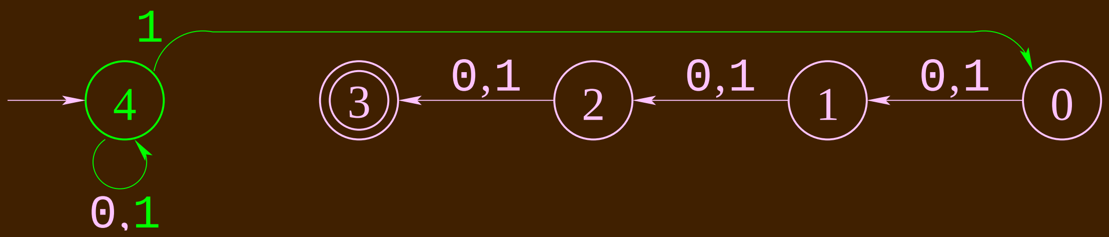
- Processing the string: `0101101` (NFA initialized)

Another NFA Example



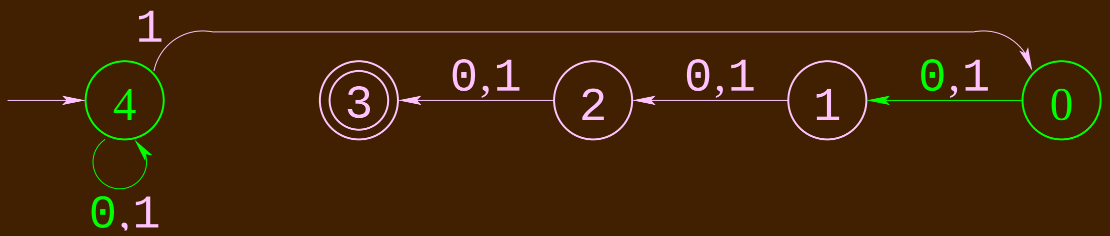
- Processing the string: 0101101 (reading the first symbol)
- Reject ☹: ϵ

Another NFA Example



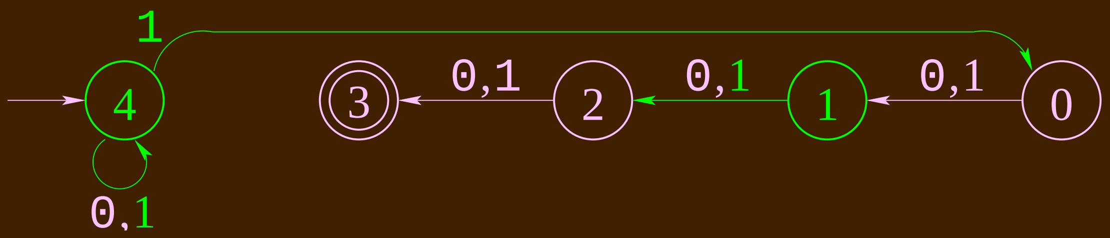
- Processing the string: `0101101` (first move done)
- Reject ☹: ϵ, `0`

Another NFA Example



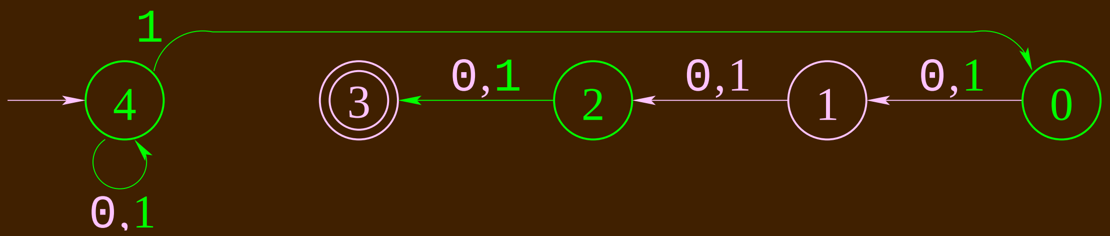
- Processing the string: `01`**`0`**`1101` (second move done, . . .)
- Reject ☹: ϵ, `0`, `01`

Another NFA Example



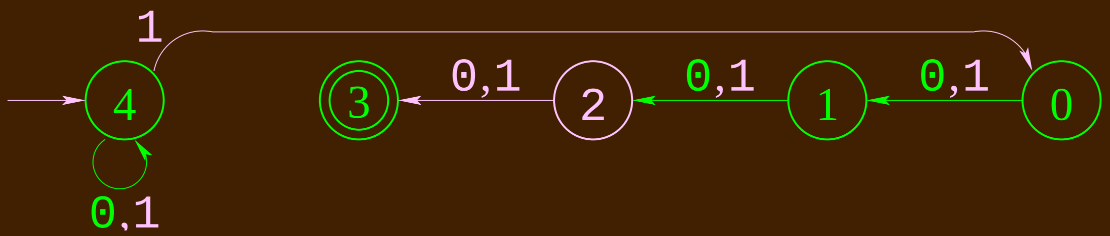
- Processing the string: `0101101`
- Reject ☹: ϵ, `0`, `01`, `010`

Another NFA Example



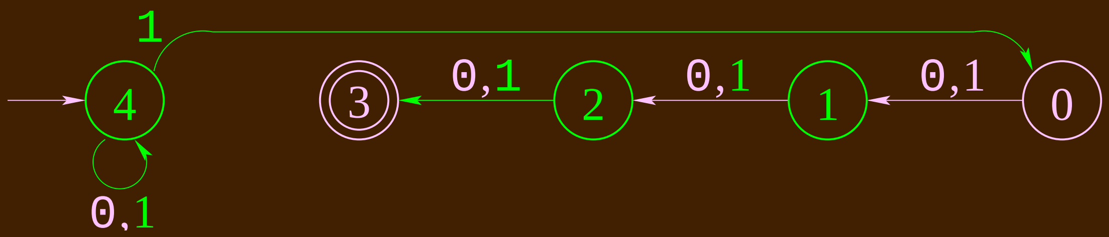
- Processing the string: `0101101`
- Reject ☹: ϵ, `0`, `01`, `010`, `0101`

Another NFA Example



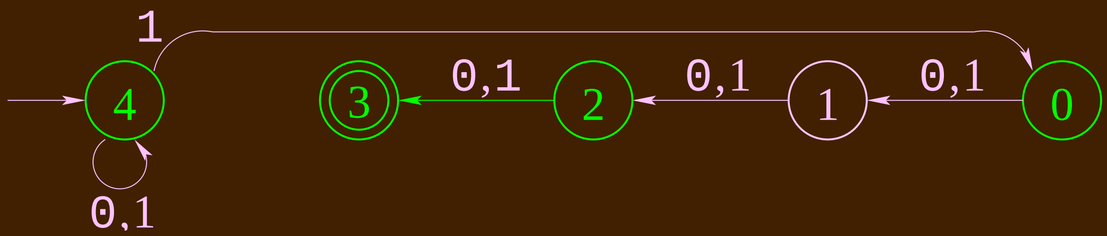
- Processing the string: `0101101`
- Reject ☹: ϵ, `0`, `01`, `010`, `0101`
- Accept ☺: `01011`

Another NFA Example



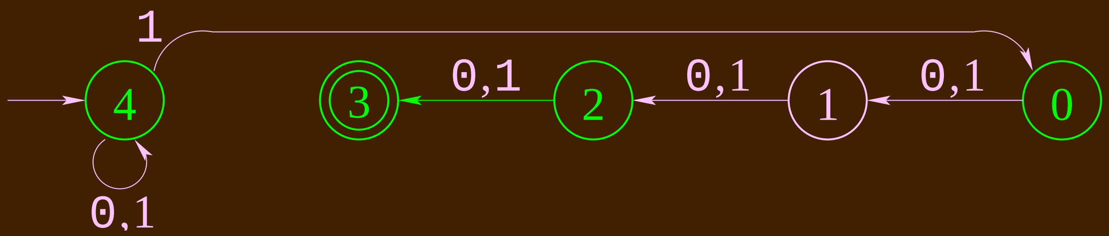
- Processing the string: `0101101`
- Reject ☹: ϵ, `0`, `01`, `010`, `0101`, `010110`
- Accept ☺: `01011`

Another NFA Example



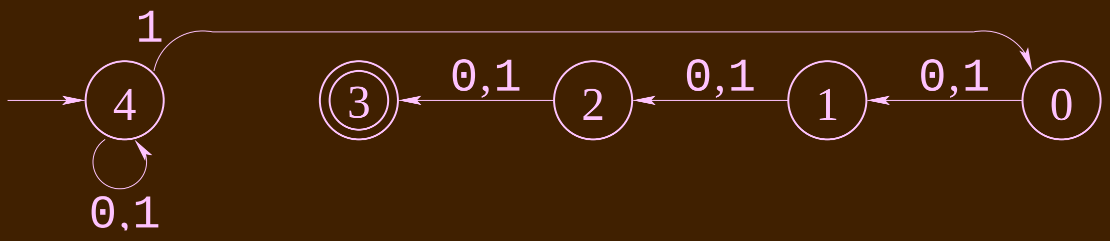
- Processing the string: `0101101` (done, string accepted)
- Reject ☹: ϵ, `0`, `01`, `010`, `0101`, `010110`
- Accept ☺: `01011`, `0101101`

Another NFA Example



- Processing the string: `0101101` (done, string accepted)
- Reject ☹: ϵ, `0`, `01`, `010`, `0101`, `010110`.
- Accept ☺: `01011`, `0101101`.
- What language does this machine accept?

Formalizing This NFA



- Let $N = (Q_N, \Sigma, \delta_N, q_{0,N}, F_N)$ be an NFA with
- $Q_N = \{0, 1, 2, 3, 4\}$;
- $\Sigma = \{0, 1\}$;
- $$\begin{array}{rcll} \delta_N(q, c) & = & \{q+1\}, & q \in \{0,1,2\}, c \in \Sigma \\ \delta_N(3, c) & = & \emptyset, & c \in \Sigma \\ \delta_N(4, 0) & = & \{4\} & \\ \delta_N(4, 1) & = & \{0, 4\}; & \end{array}$$
- $q_{0,N} = 4$;
- $F_N = \{3\}$.

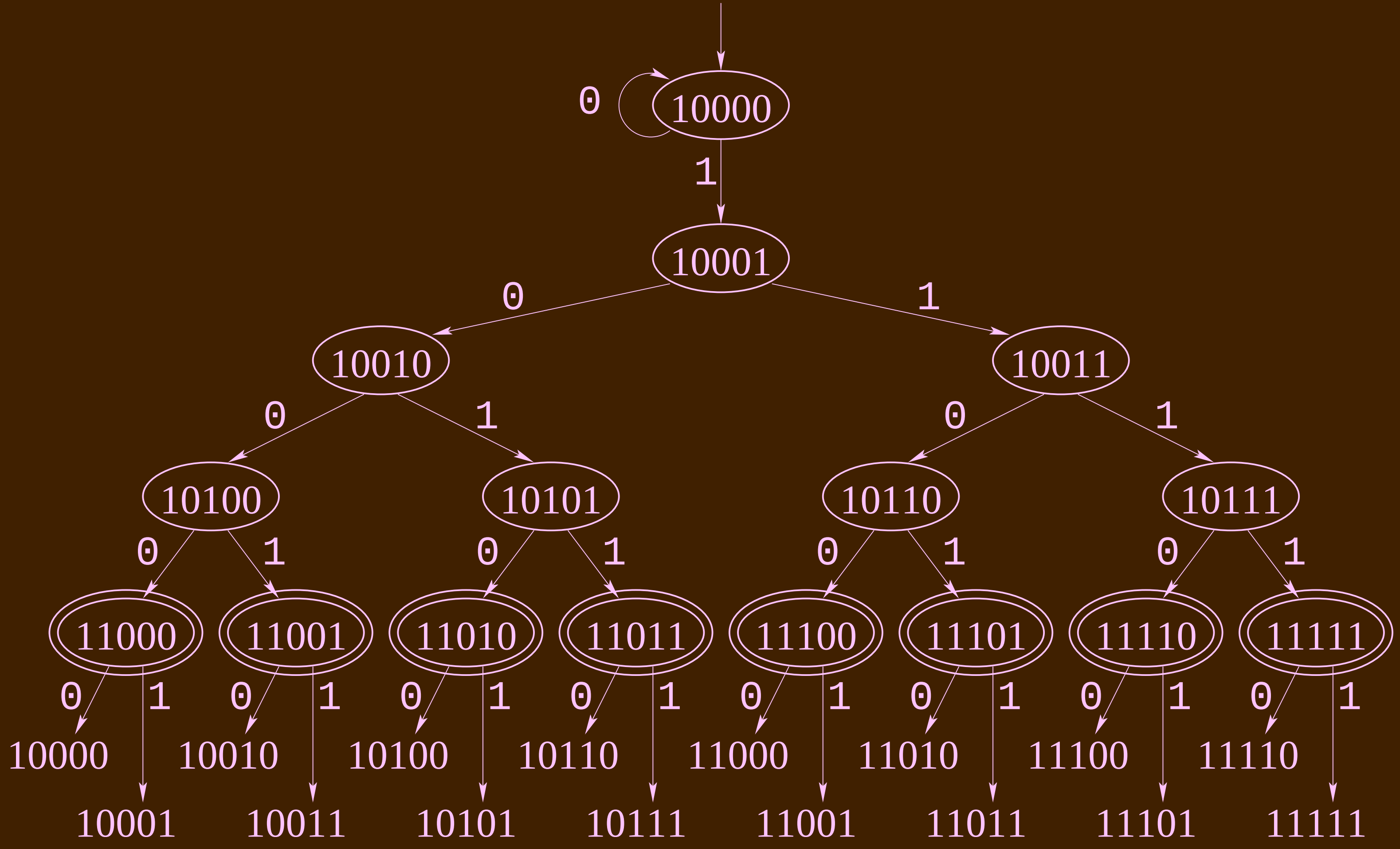
The Equivalent DFA (formal)

- First, we note that the NFA from the previous slide has no ϵ moves. Thus, $\phi(B) = B$ for any $B \subseteq Q$.
- Let $D = (Q_D, \Sigma, \delta_D, q_{0,D}, F_D)$ be a DFA with
- $Q_D = 2^{Q_N}$. We'll also treat the 32 elements of Q_D as (binary) integers. For any $B \in Q_D$,

$$B \quad \equiv \quad \sum_{k:q_k \in B} 2^k$$

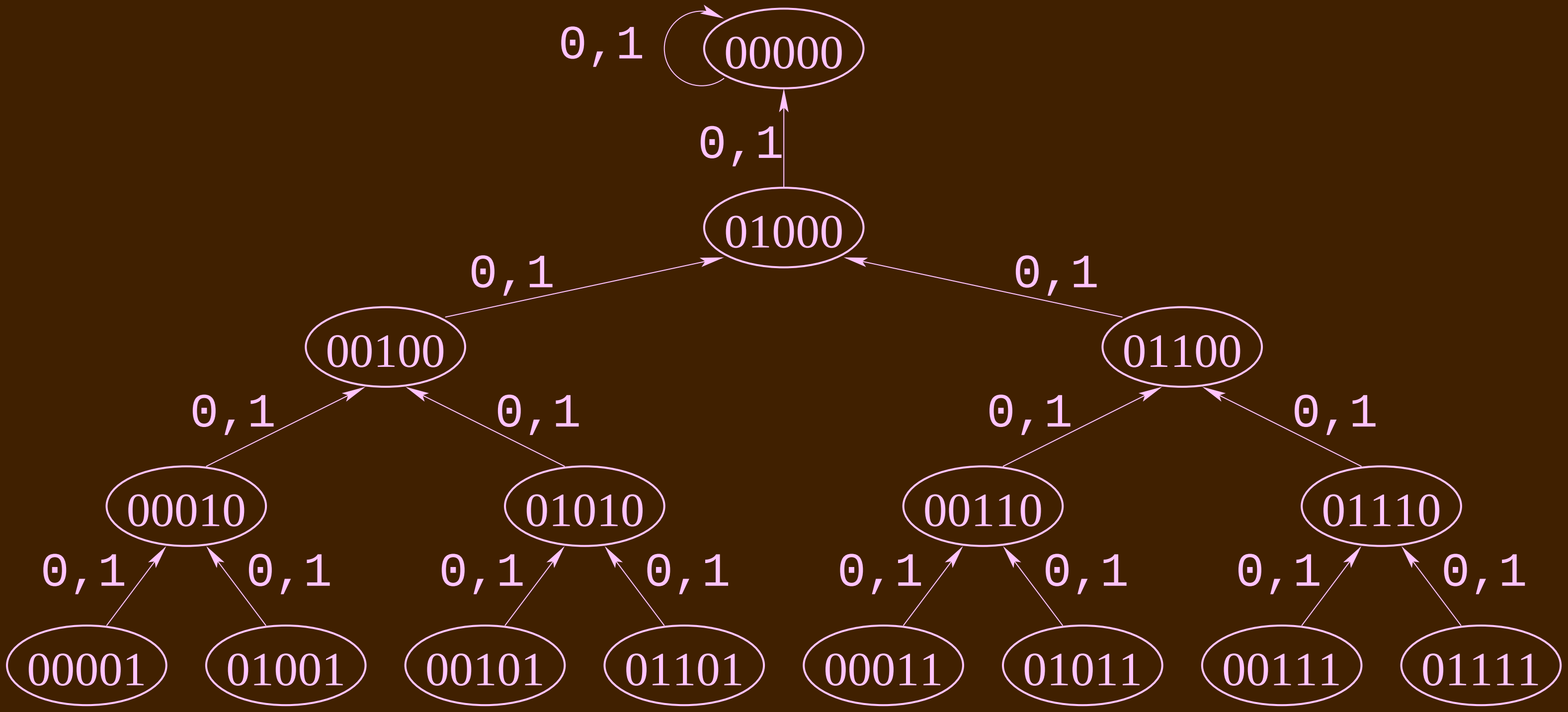
- $\Sigma = \{0, 1\}$ – the same as for the NFA.
- $$\begin{aligned} \delta_D(B, c) \quad &= \quad \phi(\textstyle\bigcup_{q \in B} \delta_N(q, c)) \\ &\equiv \quad ((2 * B) \bmod 16) + (B \text{ div } 16) * (16 + c) \end{aligned}$$
- $q_{0,D} = \{4\} \equiv 2^4 = 16$
- $F_D \equiv \{k \mid (8 \leq k < 16) \vee (24 \leq k < 31)\}$

The Equivalent DFA (diagram)



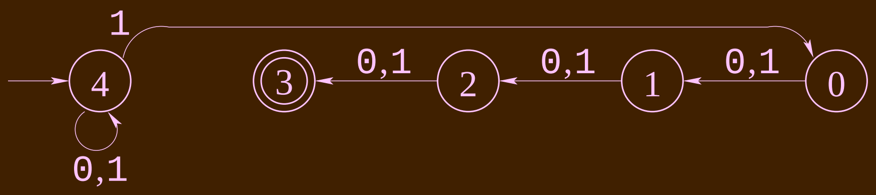
This diagram shows the states that include state 4 from the NFA (i.e. states with binary values ≥ 16).

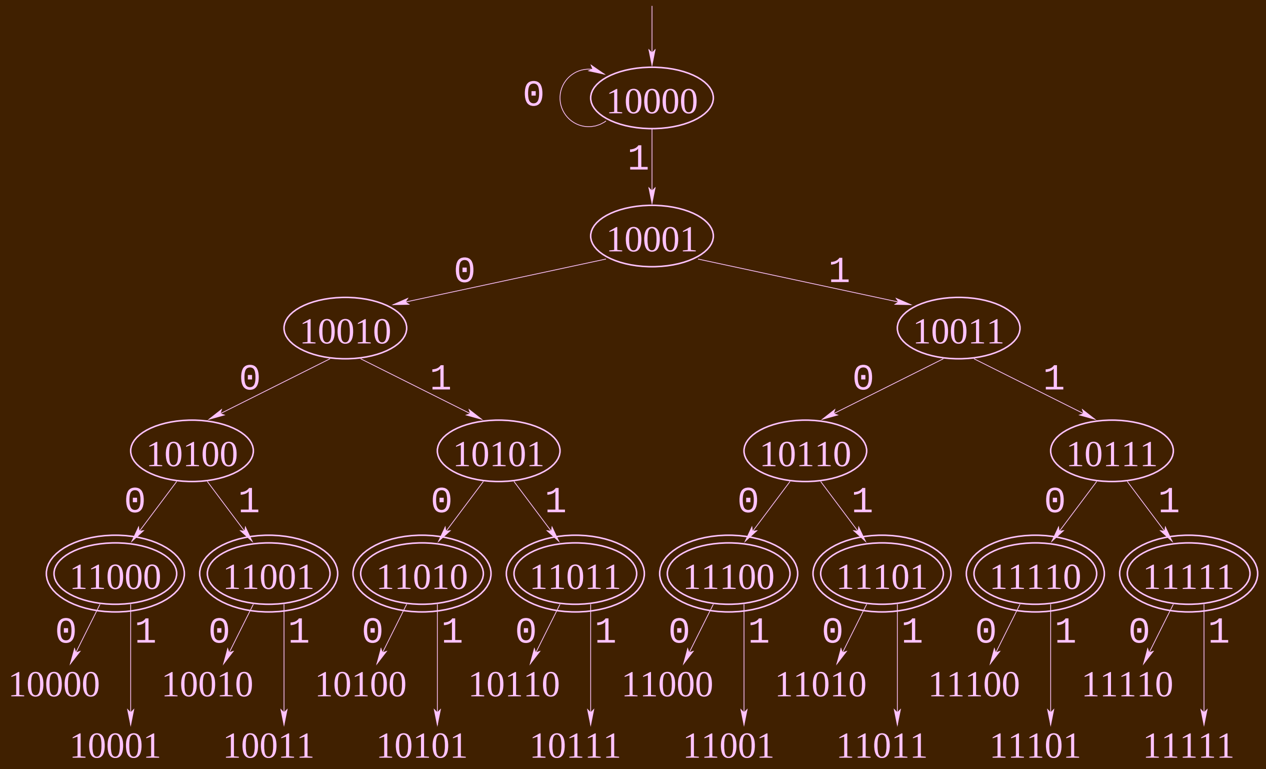
The Equivalent DFA (diagram)



This diagram shows the states that don't include state 4 from the NFA (i.e. states with binary values < 16).

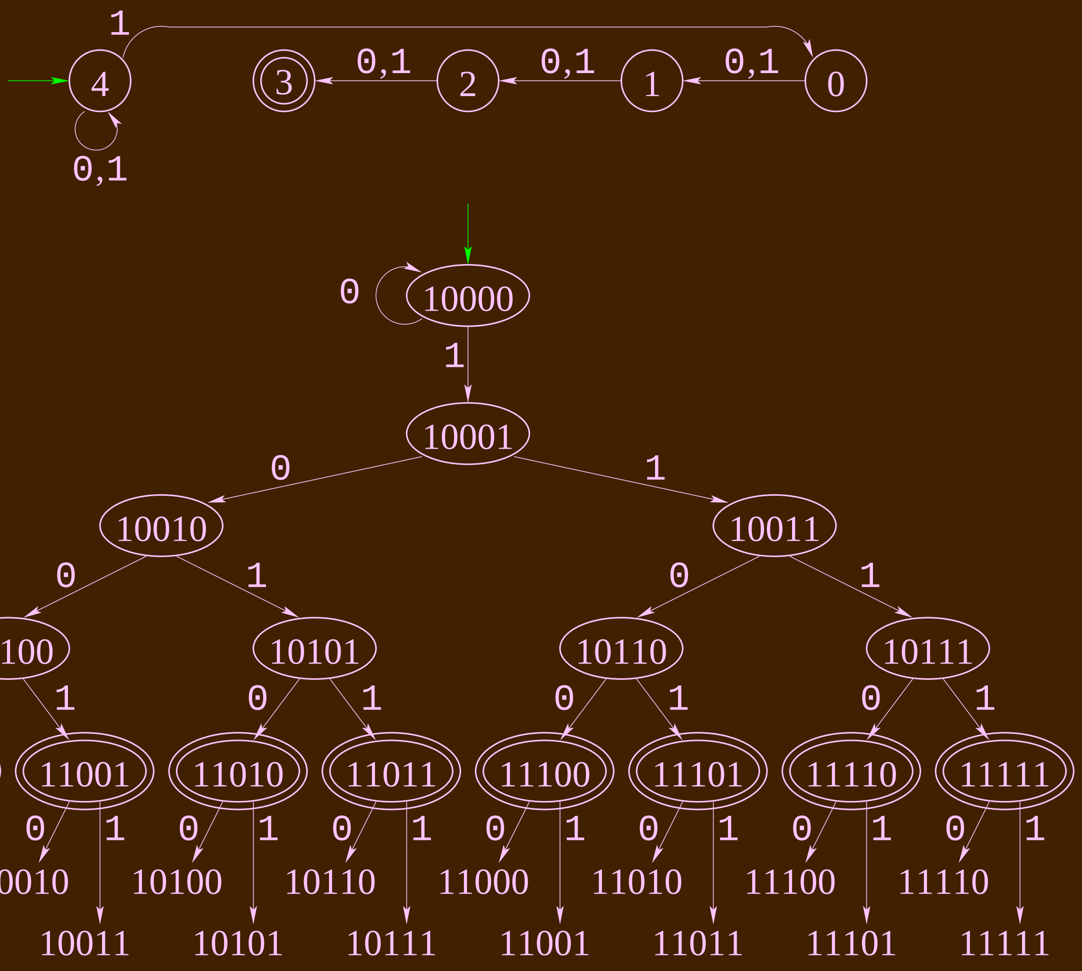
Simulating the NFA with the DFA





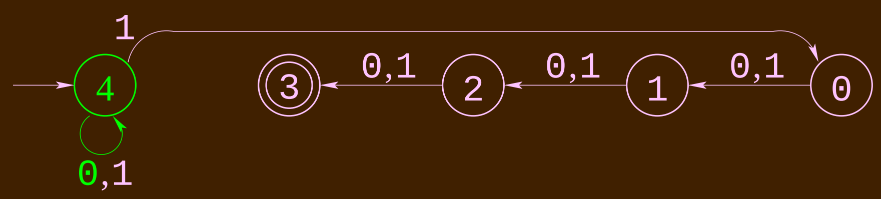
- Processing the string: `0101101`

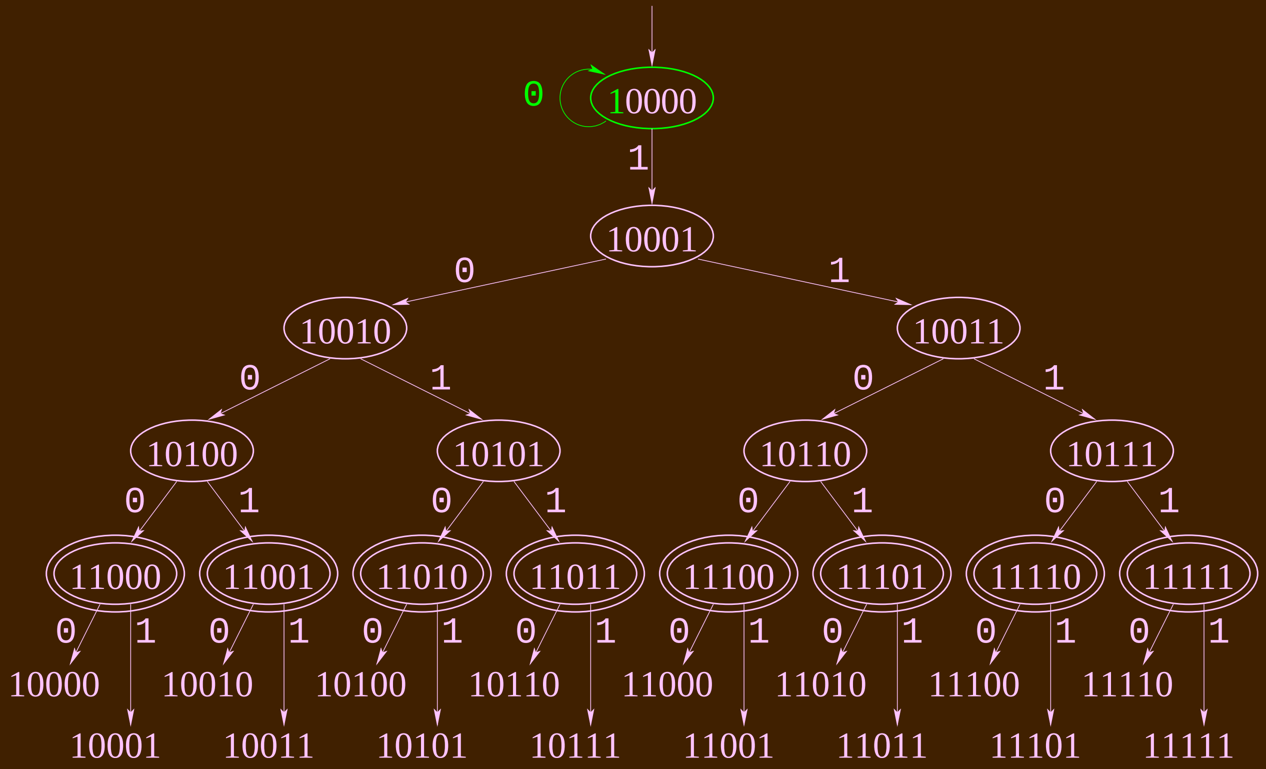
Simulating the NFA with the DFA



- Processing the string: `0101101` (NFA initialized)

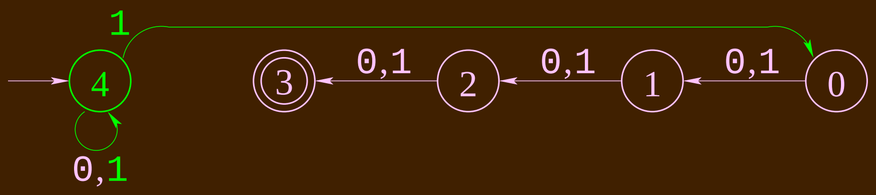
Simulating the NFA with the DFA

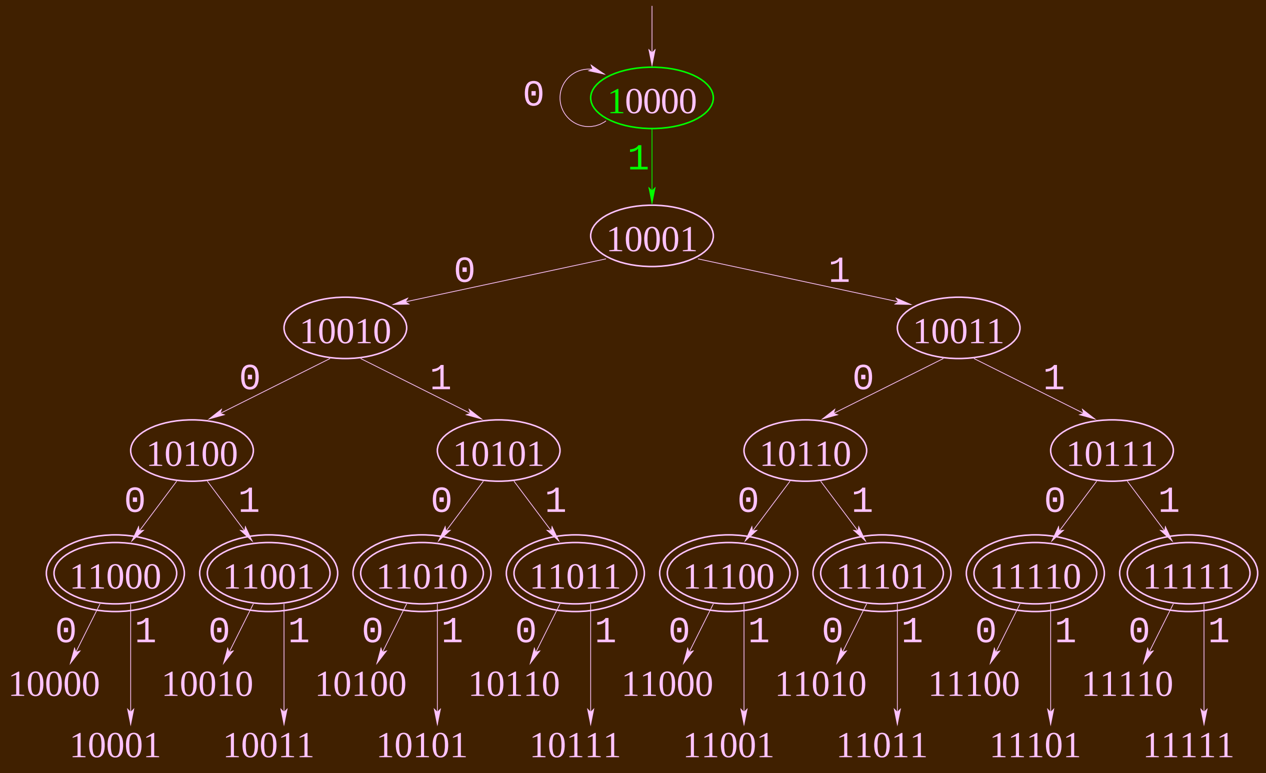




- Processing the string: `0101101` (reading the first symbol)
- Reject: ϵ.

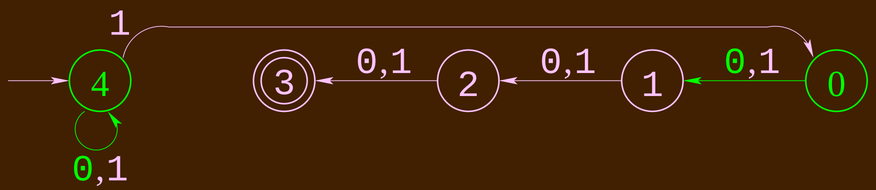
Simulating the NFA with the DFA

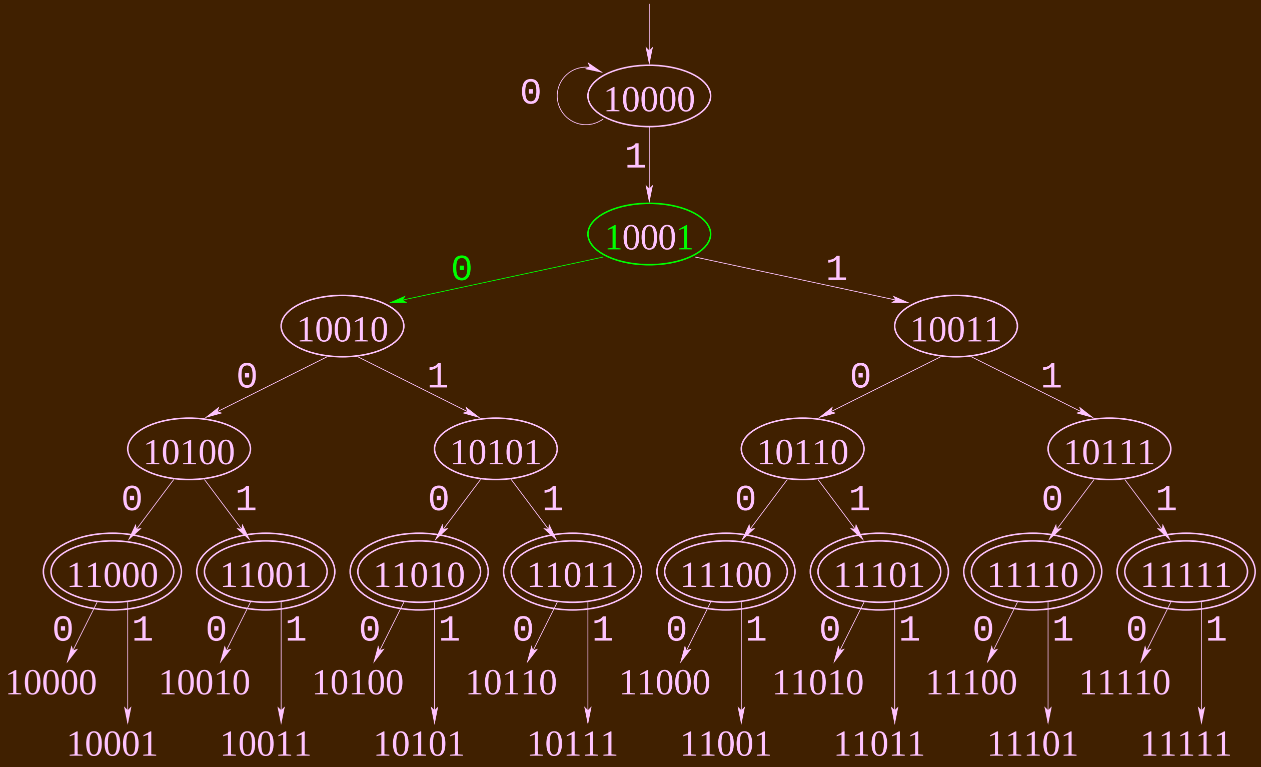




- Processing the string: `0101101` (first move done)
- Reject: `0`.

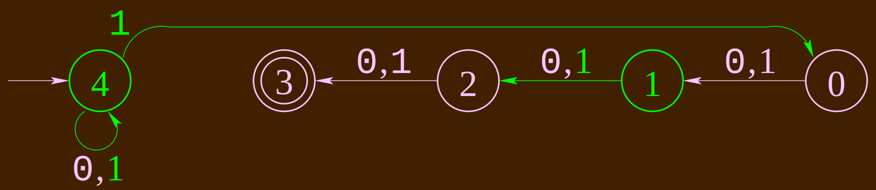
Simulating the NFA with the DFA

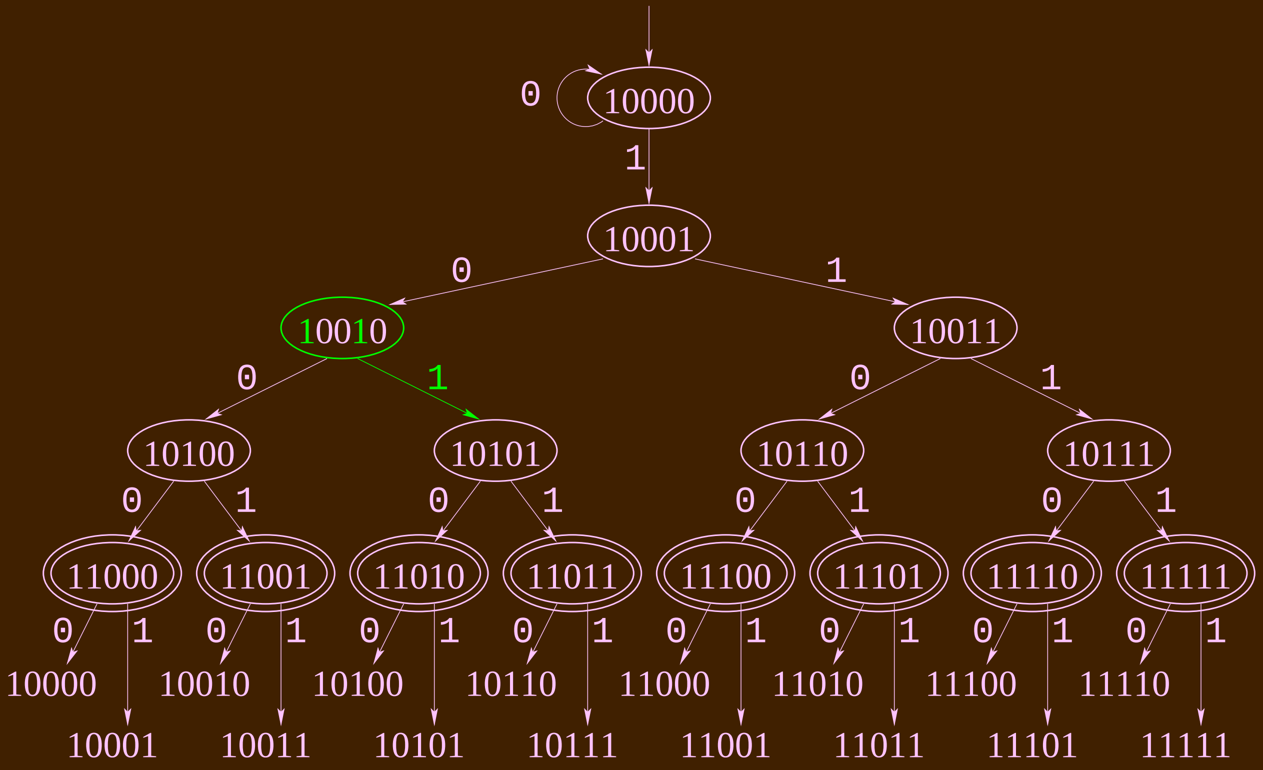




- Processing the string: `0101101` (second move done, . . .)
- Reject: `01`.

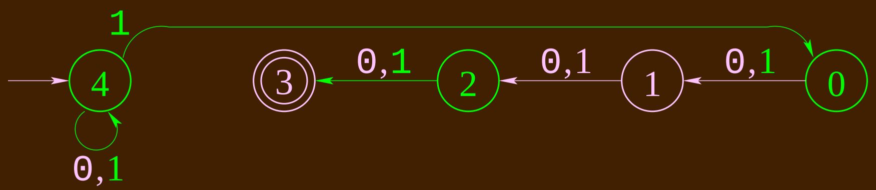
Simulating the NFA with the DFA

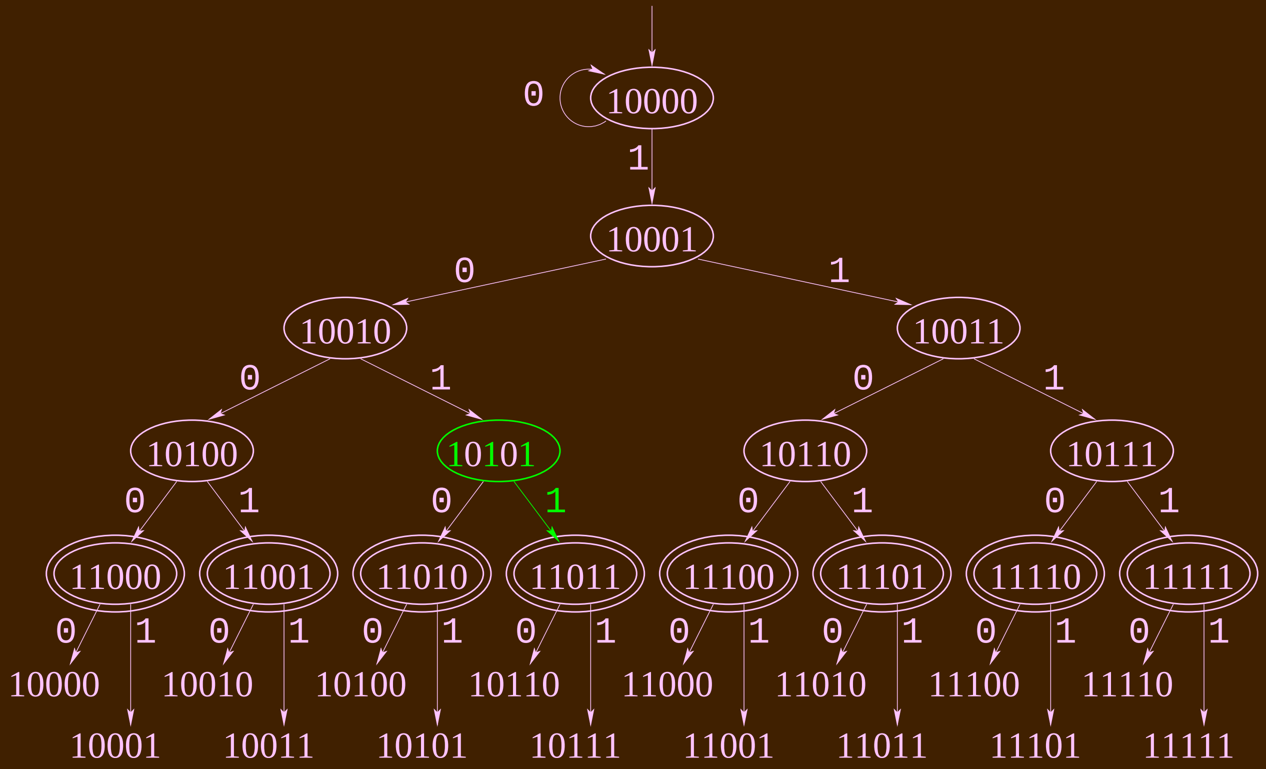




- Processing the string: `0101101`
- Reject: `010`.

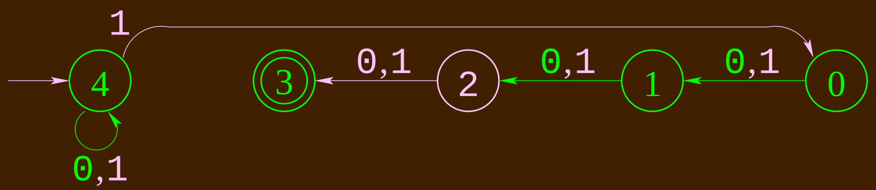
Simulating the NFA with the DFA

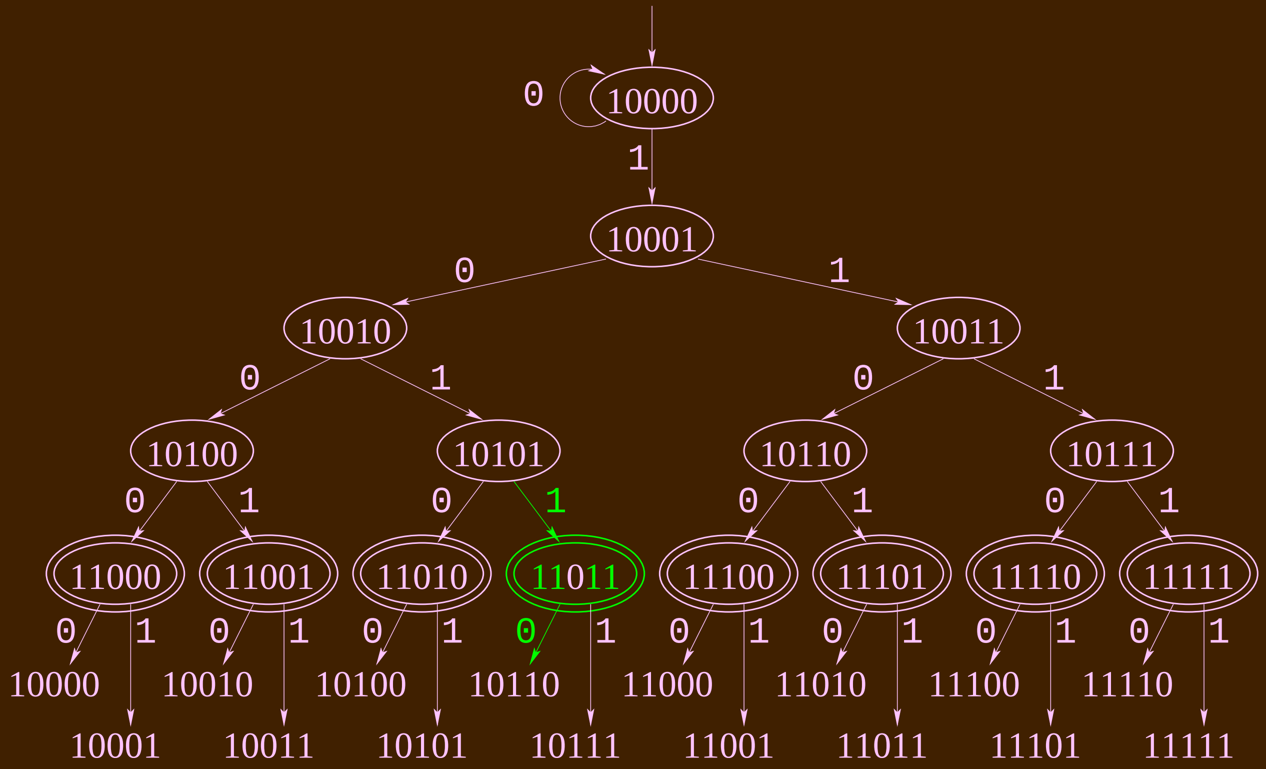




- Processing the string: `0101101`
- Reject: `0101`.

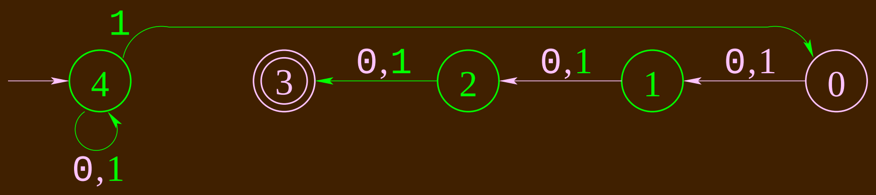
Simulating the NFA with the DFA

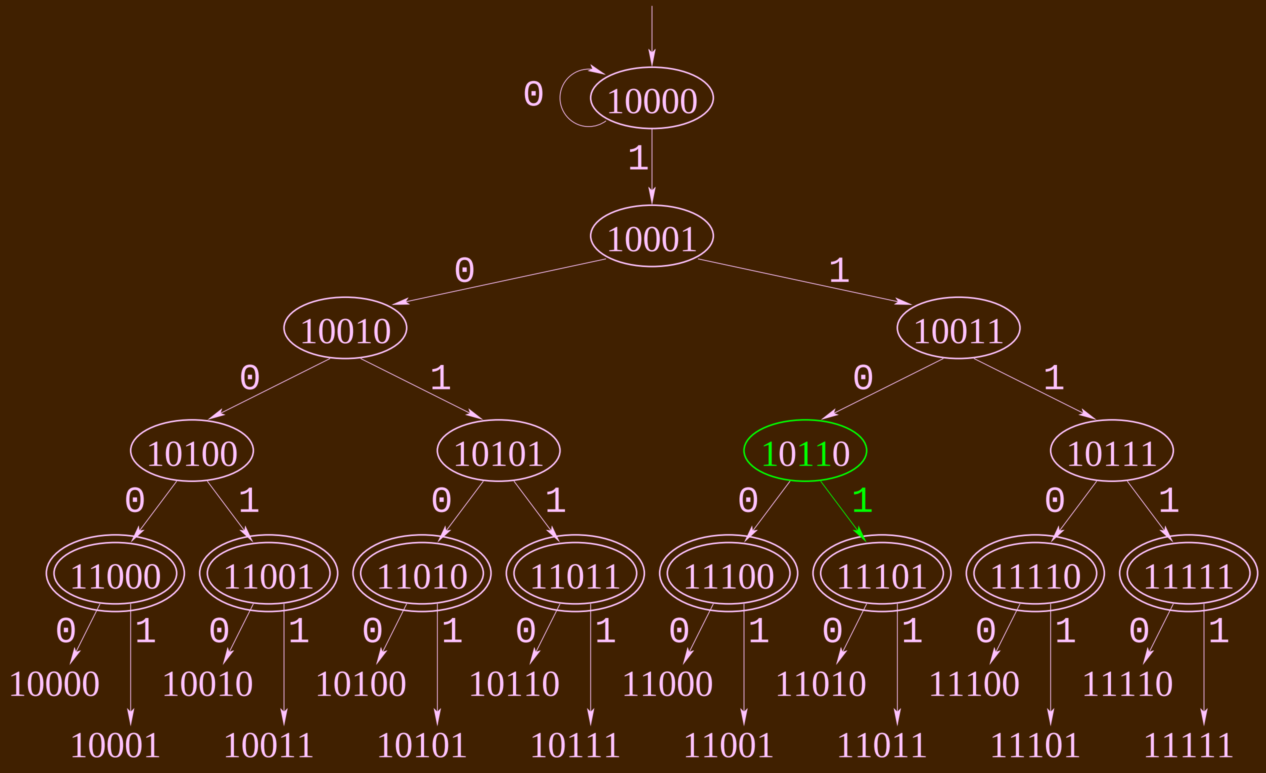




- Processing the string: `0101101`
- Accept: `01011`.

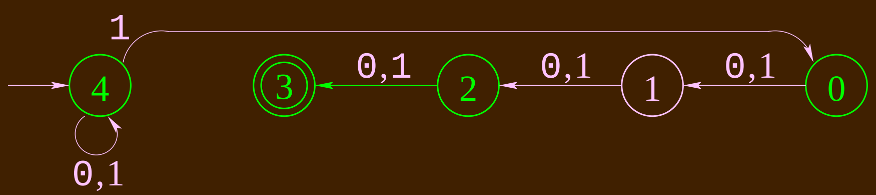
Simulating the NFA with the DFA

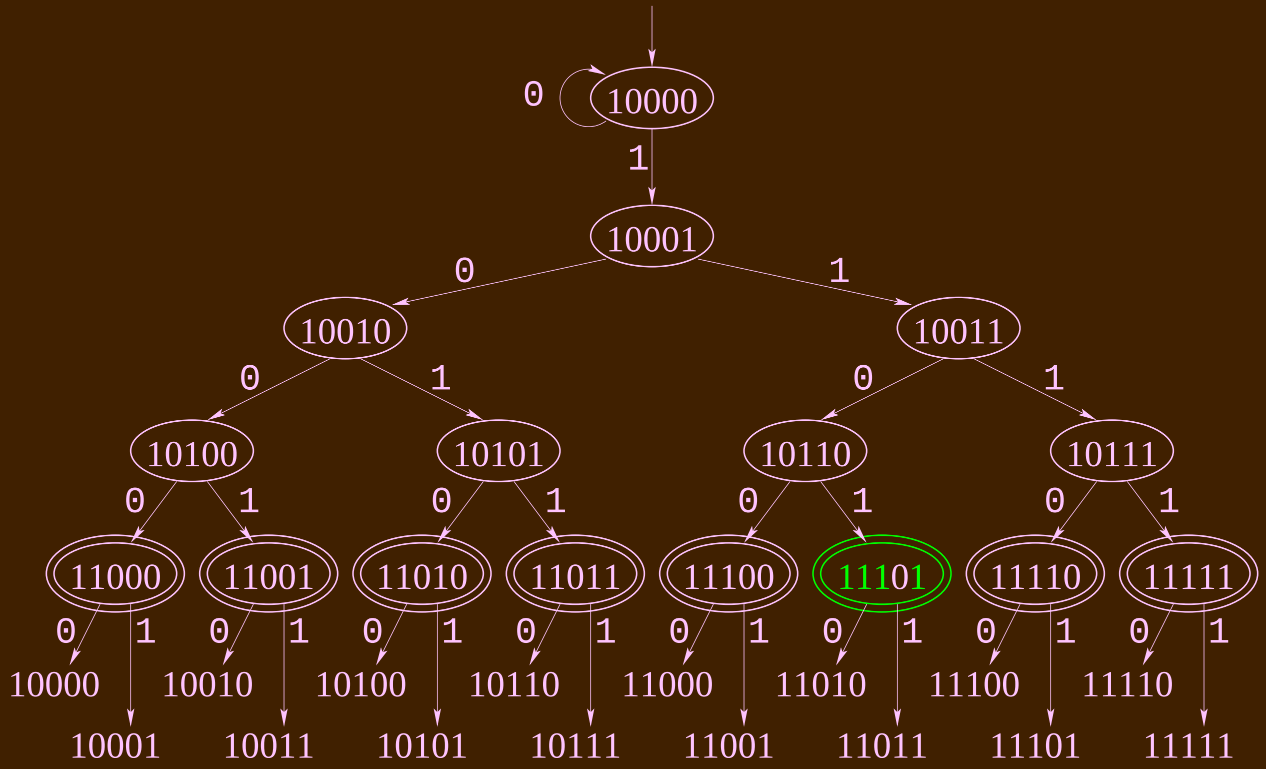




- Processing the string: `0101101`
- Reject: `010110`.

Simulating the NFA with the DFA





- Processing the string: `0101101` (done, string accepted)
- Accept: `0101101`.

Regular Madlibs

Once upon a $\underset{\text{noun}}{\underline{\hspace{2cm}}}$, there was a $\underset{\text{noun}}{\underline{\hspace{2cm}}}$ that $\underset{\text{past tense verb}}{\underline{\hspace{3.5cm}}}$

$\underset{\text{zero or more adjectives}}{\underline{\hspace{5cm}}}$ $\underset{\text{plural noun}}{\underline{\hspace{2.5cm}}}$.

- Here are some examples of nouns: `avocado`, `beach`, `caterpillar`, . . . `zombie`.
- Let $\Sigma = \{$`a`, `b`, . . . `z`, `A`, . . . `z`, *<space>*, *<period>*, *<comma>*$\}$.
- Clearly, the language $\{$`a`$\}$ a regular language. Likewise $\{$`v`$\}$ is a regular language.
- Regular languages are closed under concatenation, therefore $\{$`avocado`$\}$, $\{$`beach`$\}$, . . . and $\{$`zombie`$\}$ are regular languages.
- Regular languages are closed under union, therefore $\{$`avocado`, `beach`, . . . `zombie`$\}$ is a regular language.

[See www.madlibs.org].

More Madlibs

Once upon a $\underset{\text{noun}}{\underline{\hspace{5em}}}$, there was a $\underset{\text{noun}}{\underline{\hspace{5em}}}$ that $\underset{\text{past tense verb}}{\underline{\hspace{8em}}}$

$\underset{\text{zero or more adjectives}}{\underline{\hspace{12em}}}$ $\underset{\text{plural noun}}{\underline{\hspace{6em}}}$.

- Let $noun = \texttt{avocado} \cup \texttt{beach} \cup \texttt{caterpillar} \cup \texttt{zombie}$.
- Let $pluralNoun = noun\,\texttt{s}$.
- Let $verb = \texttt{add} \cup \texttt{compile} \cup \texttt{eat} \cup \texttt{sing} \cup \texttt{swim}$.
- Let $pastVerb = verb\,\texttt{ed}$.
- Let $adjective = \texttt{big} \cup \texttt{cold} \cup \texttt{furry} \cup \texttt{insipid} \cup \texttt{yellow}$.
- Now, our Madlib$^{\text{TM}}$ is

  `Once upon a` *`noun`*`, there was a` *`noun`* `that`
  `pastVerb` $($*`adjective`*$)^*$ *`pluralNoun`*`.`

Regular Expressions

Our Madlib^{TM}s are examples of regular expressions. In the next lecture, we will:

- Give a formal defintion of regular expressions.
- Show that the set of languages describe by regular expressions is the set of regular languages:
  - We'll show that for every regular expression, there is an NFA that recognizes the same language. Having just shown the equivalence of NFAs and DFAs, this establishes that the languages described by regular expressions are regular.
  - We'll then show that any language recognized by a DFA can be described by a regular expression. This shows that all regular languages can be described by regular expressions and completes the proof.