Outline

- **Lexical Analysis**
  - Regular Expressions
  - Finite State Automata
  - DFSA $\Leftrightarrow$ NFSA $\Leftrightarrow$ RE Conversions
- Syntactical Analysis
  - Context Free Languages
  - Parsing (next two lectures)

Describing a Language

- Layered structure of language definition
  - The alphabet - basic symbols
  - Lexical structure - words
  - Syntactic structure - statements
  - Semantics - meaning
  - Today's topic: lexical and syntactic structures

Formal Languages

- A language is a set of strings
- Dual Approaches
  - Generation (grammar)
  - Recognition (parser)
- Formal relations and mechanical conversions

Lexical Analysis

- Source program text → Tokens

- Examples of Tokens
  - Operators = + - > ({ := == <>
  - Keywords if while for int double
  - Numeric literals 43 6.035 -3.6e10 0x13F3A
  - Character literals 'a' '~' '\"'
  - String literals "6.891" "Fall 98" "\"\" = empty"
- Examples of non-tokens
  - White space space(' ') tab('\t') end-of-line('\n')
  - Comments /*this is not a token*/

A Lexical Analyzer in Action

f o r v a r 1 = 1 0 v a r 1 < =

A Lexical Analyzer in Action

f o r v a r 1 = 1 0 v a r 1 < =

for ID("var1") eq_op Num(10) ID("var1") leq_op

- Partition the texts into a sequence of tokens
- Attach attributes to tokens
- Eliminate white space and comments

Lexical Analyzer needs to….

- Identify the type of tokens
  - 6036 Num(6035)
  - X6035 ID("X6035")
- Tokens in different languages
  - FORTRAN DO I=1,10
  - C++ for(int i=1; i<= 10; i++)
  - R/S-plus for (i in 1:10)
- Too hard to do in an ad-hoc basis

We need a language to describe lexical structure!!

Regular Expressions

- A language to define lexical structure
  - describes how to generate tokens of a particular type
  - a regular expression == a set of strings

- Defined hierarchically (surprised?)

The meaning of a Regular Expression

- Expression = Set of strings from an alphabet $\sum$
  - Let S(e) be the set of strings for reg-exp e
  - S(e) is defined hierarchically
- Base symbols:
  - Characters c $\in \sum$ S(c) = { c }
  - ε - empty string S(ε) = {""}
- Derived Expressions:
  - concatenation $e_1\ e_2$ $S(e_1\ e_2)=\{xy : x \in S(e_1)\ \&\ y \in S(e_2)\}$
  - choice $e_1|e_2$ $S(e_1|e_2)= S(e_1) \cup S(e_2)$
  - Kleen closure e^* $S(e^*) = S(\varepsilon) \cup S(e) \cup S(ee) \cup S(eee) \cup \ldots$

Examples

- Let $\Sigma = \{0, 1\}$

- 1 → { 1 }
- 0 1 → {01}
- 0 | 1 → {0, 1}
- 0(0|1) → {00, 01}
- 0(0|1)(1| ε) → {001, 011, 00, 01}
- 0(01)* → {0, 001, 00101, 0010101,...}

Examples of Regular Languages

- Σ = { 0, 1, . }
  - (0|1)*.(0|1)* - Binary floating point numbers
  - (00)* - even-length all-zero strings
  - 1*(01*01*)* - strings with even number of zeros
- Σ = { a,b,c, 0, 1, 2 }
  - (a|b|c)(a|b|c|0|1|2)* - alphanumeric identifiers
  - (0|1|2)* - trinary numbers

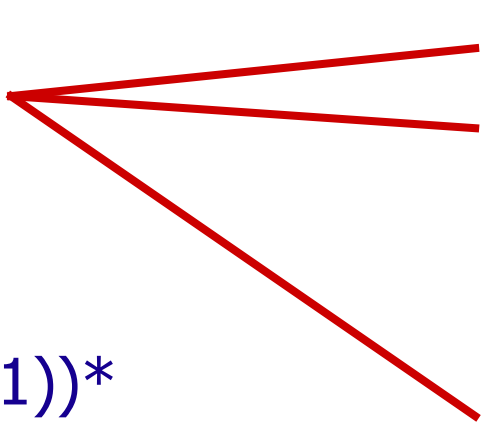
Match Strings and Regular Expressions

1. 0(0|1)*0
2. ((ε|0)1*)*
3. ((0|1)0(0|1))*

a. 000000
b. 01010
c. 010101
d. 101010
e. 001100

Match and Create the Regular Expressions

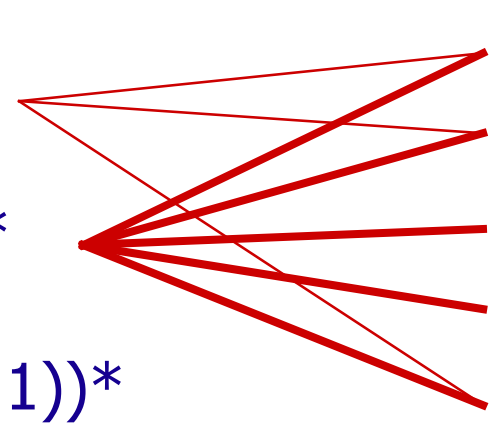
1. 0(0|1)*0
2. ((ε|0)1*)*
3. ((0|1)0(0|1))*



a. 000000
b. 01010
c. 010101
d. 101010
e. 001100

Match and Create the Regular Expressions

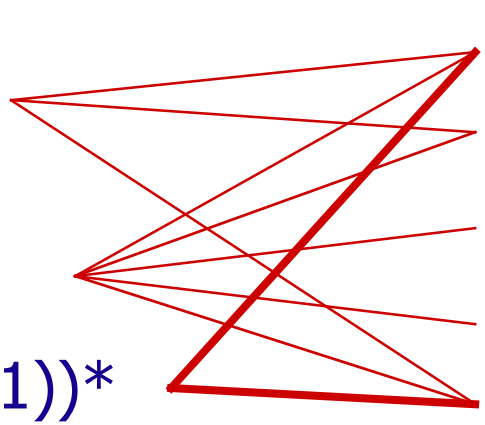
1. 0(0|1)*0
2. ((ε|0)1*)*
3. ((0|1)0(0|1))*



a. 000000
b. 01010
c. 010101
d. 101010
e. 001100

Match and Create the Regular Expressions

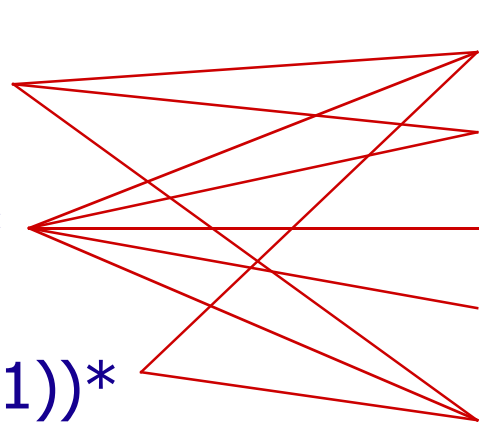
1. 0(0|1)*0
2. ((ε|0)1*)*
3. ((0|1)0(0|1))*



a. 000000
b. 01010
c. 010101
d. 101010
e. 001100

Match and Create the Regular Expressions

1. 0(0|1)*0
2. ((ε|0)1*)*
3. ((0|1)0(0|1))*



a. 000000
b. 01010
c. 010101
d. 101010
e. 001100

- All strings of 0's and 1's that does not contain the substring 011

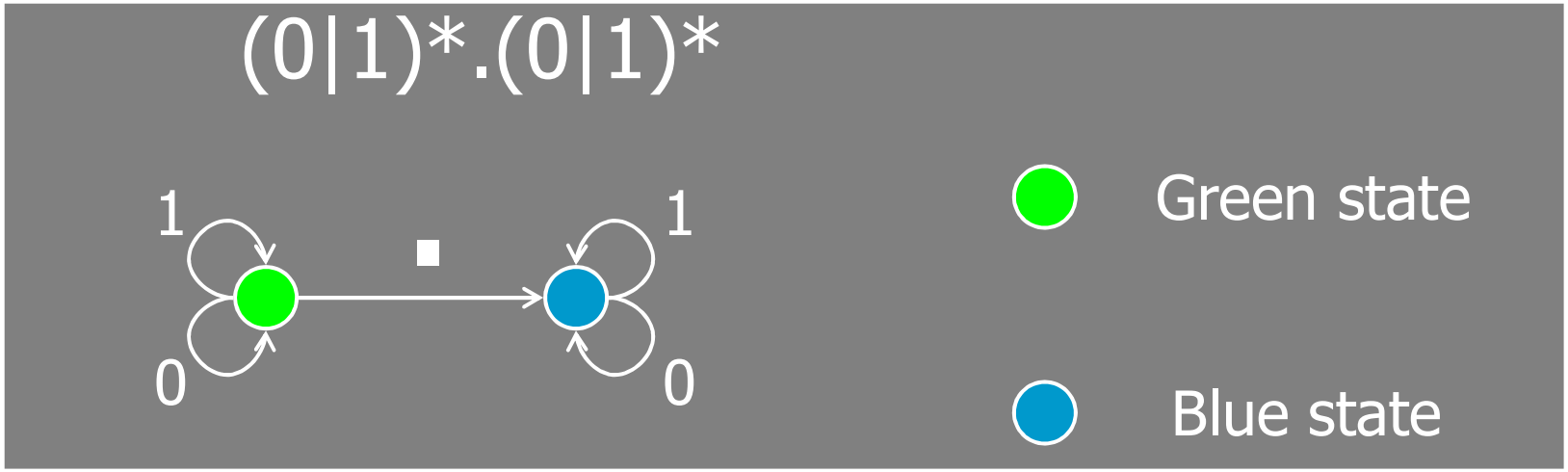
Generation Versus Recognition

- Regular Expressions
  - Are a great language for describing a set of strings
  - But we don't want to generate tokens
    - we want to recognize them

- For that we need a different formalism

Finite-State Automata

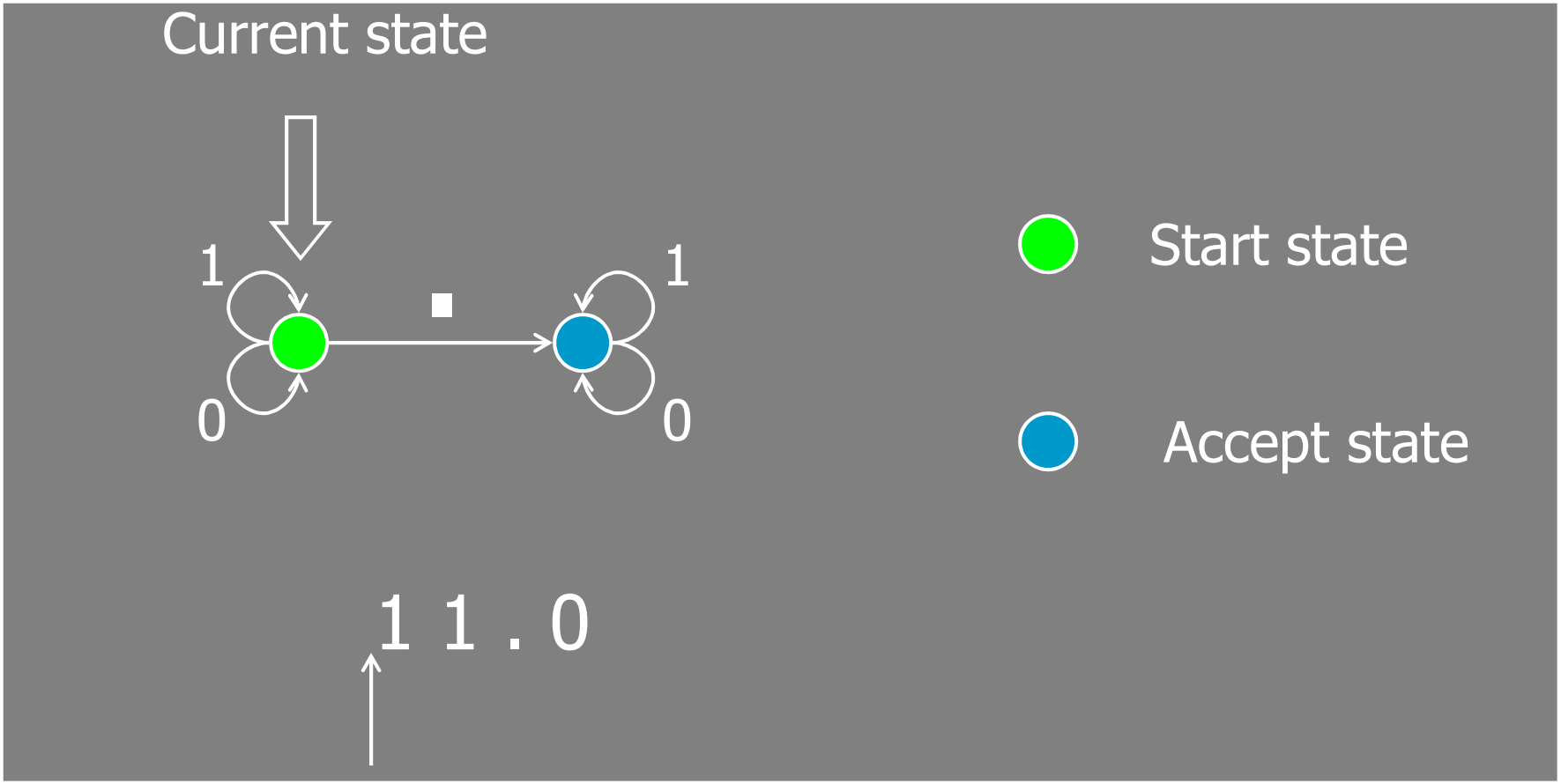
- Alphabet $\sum$
- Set of states with initial and accept states
- Transitions between states labeled with letters



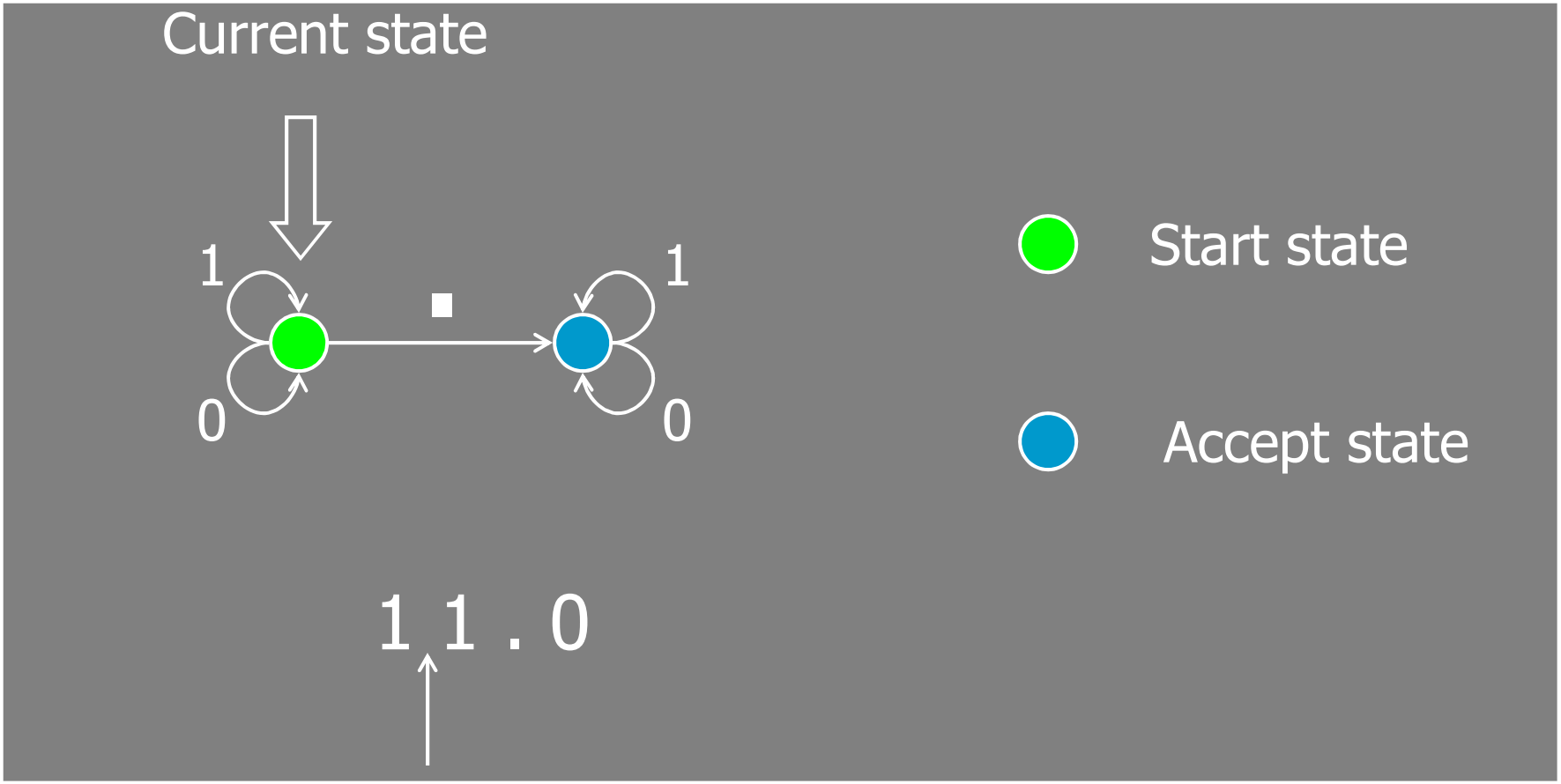
Automaton

- String recognition
  - Start with start state and first letter of string
  - At each step, match current letter against a transition whose label is same as letter
  - Continue until reach end of string or match fails
  - If end in accept state, automaton accepts string
- The language of automaton is the set of strings it accepts

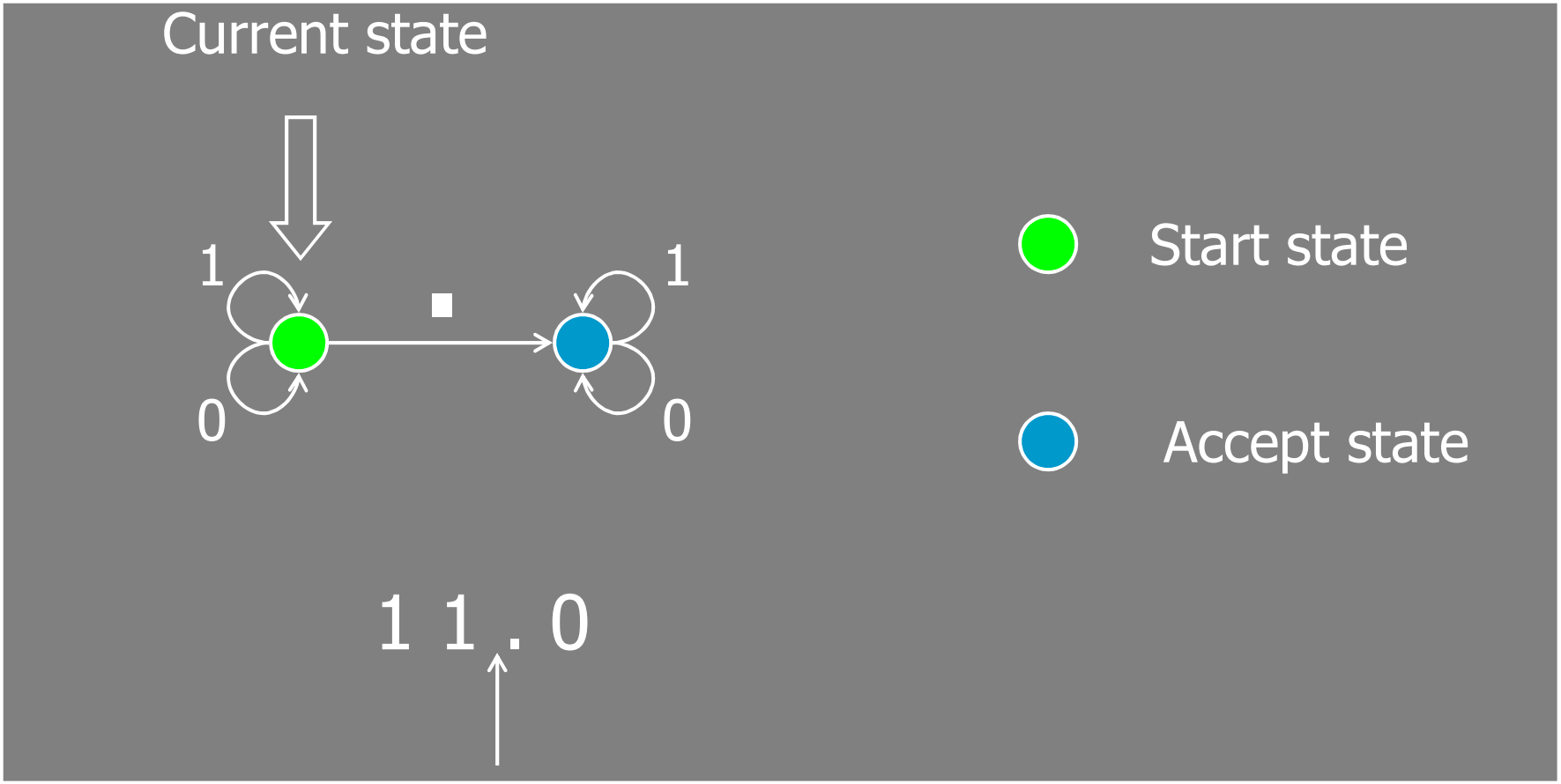
Example



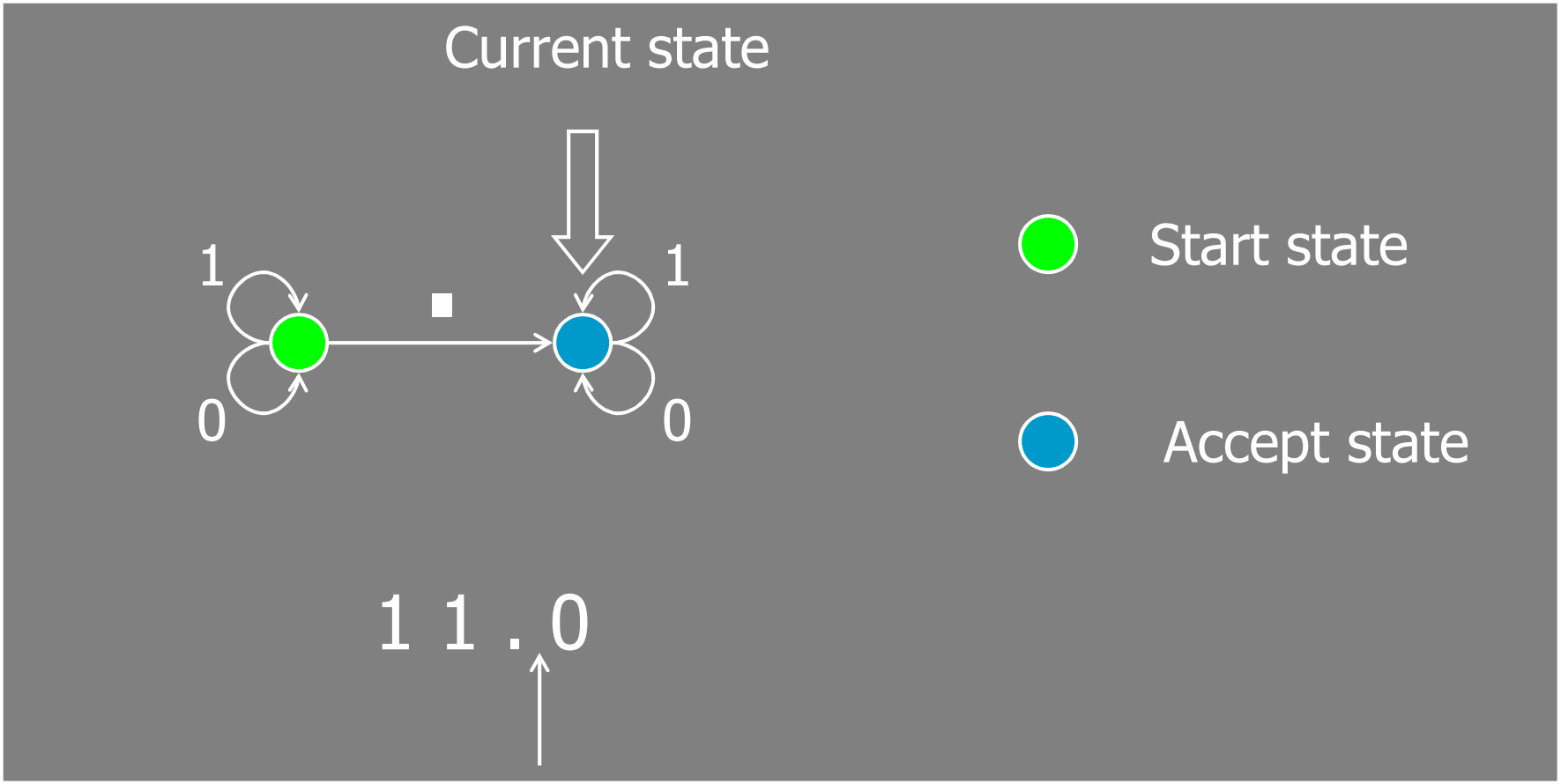
Example



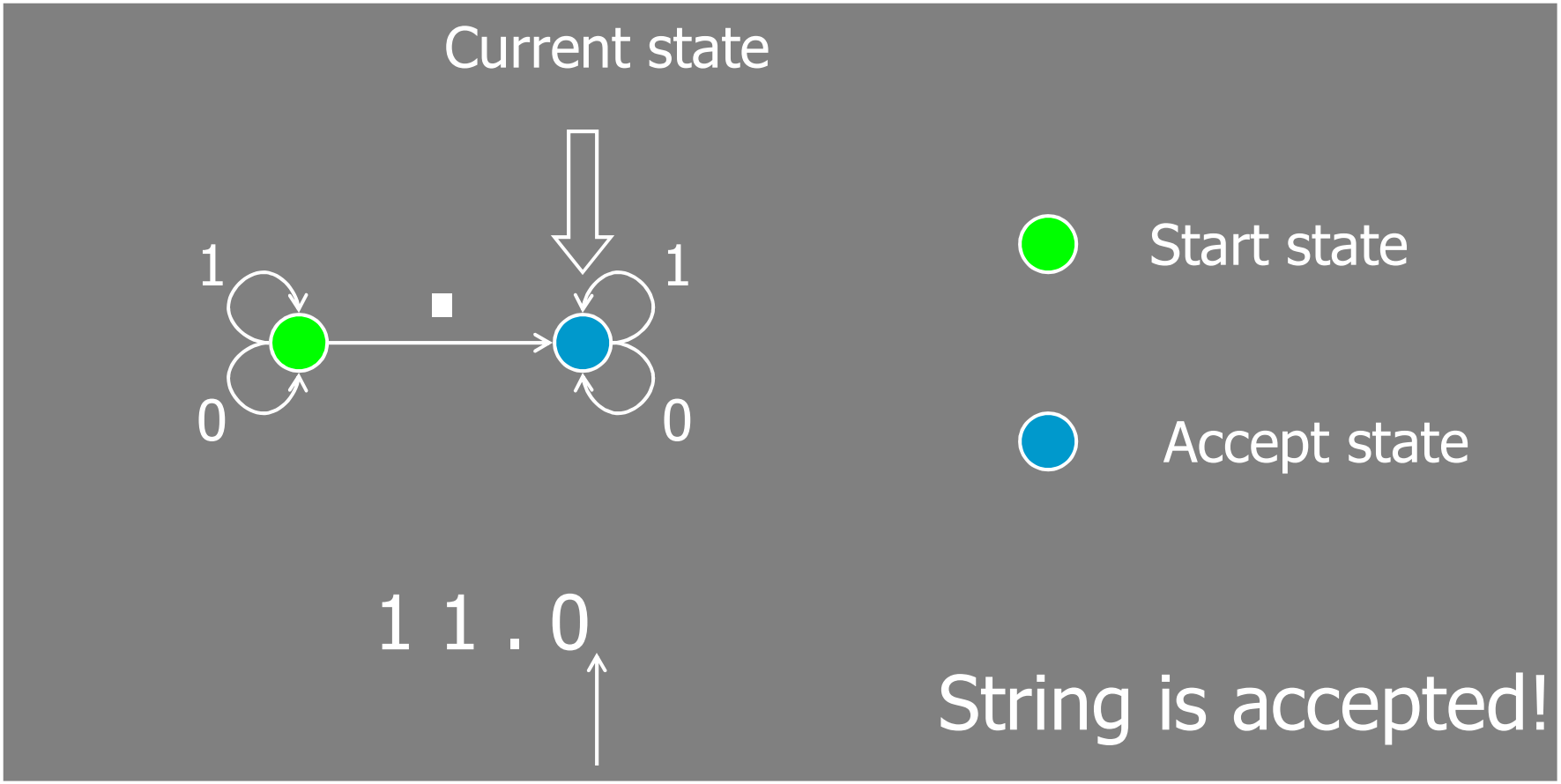
Example



Example



Example


Current state
1
1
0
0
Start state
Accept state
1 1 . 0
String is accepted!

Generation Versus Recognition

- Syntactically a language is a set of strings
- Regular expressions
  - Defining a language by composition
- Automata
  - Defining a language by implementation
    - Determining whether a string is in the language or not
  - Theoretically equivalent (for regular expressions and automata)

Can we compile the Regular Expression Language into an Automata Implementation?

Yes we can!