Flex Intro

Tutorial - 2

What is Flex?

How to install FLEX?

- Use Linux (or any other Unix-based OS)
  - **`$ sudo apt-get update`**
  - **`$ sudo apt install gcc`**
  - **`$ sudo apt install flex`**
- Windows?
  - Use **VirtualBox** or **Ubuntu-Terminal**
  - SSH to CPSC Linux Server (**Already has Flex**)

How to use FLEX?

- Prepare **`input.flex`** (or **`.lex`**)
- Run Flex with the input:
  - **`$ flex input.flex`**
- You should see **`lex.yy.c`** in the same directory. Compile it:
  - **`$ gcc lex.yy.c -lfl -o scanner`**
- Run the generated executable:
  - **`$ ./scanner`**

How to create input file?

- You should follow:
  - **Flex** conventions for Regular Expressions
  - **Flex** input file format

Conventions for Regular Expressions

- Various operations help to define specific regular expressions

Pattern	Meaning
a	the character *a*
“ab”	the character sequence *ab*, when it is a metacharacter
\a	the character *a* when it is a metacharacter
a*	zero or more repetition of *a*
a+	one or more repetition of *a*
a?	an optional *a*
a|b	*a* or *b*
(a)	*a* itself
[abc]	any of the characters *a*, *b*, or *c*
[a-d]	any of the characters *a*, *b*, *c*, or *d*
[^ab]	any character except *a* or *b*
.	any character except a *newline*
{xxx}	the regular expression that the name *xxx* represents

Input file format

- **Declarations:**
  The C code that must be inserted external to any function
- **Definitions:**
  Names of the regular Expressions

```
%{
Declarations
%}
Definitions
%%
Rules
%%
User subroutines
```

Flex input file format

Input file format

- **Rules:**

  Define the action of the scanner for each match case with the regular expressions from the definition section

- **User subroutines:**

  Any auxiliary routines or the main program

```
%{
Declarations
%}
Definitions
%%
Rules
%%
User subroutines
```

Flex input file format

Example `empty.flex`

- What happens if there is no rules in the input file?

```
%{
/*This is the scanner you get for a Flex input
without any rules*/
%}

%%

%%
int main(){
        printf("Before calling yylex()\n");
        yylex();
        printf("After calling yylex()\n");
        return 0;
}
```

Flex input file

Example PLS1.flex

- What happens if the input is “**a**”?
- What happens if the input is “**b**”?
- What happens if the input is “**ab**”?

```
%{
/*This is a simple scanner to demonstrate the
principle of longest substring*/
%}
A       a
B       b
%%
{A}     {printf(“This is character a.\n”);}
{B}     {printf(“This is character b.\n”);}
%%
int Main()
{
        yylex();
        return 0;
}
```

Flex input file

Example PLS2.flex

- What happens if the input is "**ab**"?
- This is the principle of longest substring.

```
%{
/*This is a simple scanner to demonstrate the
principle of longest substring*/
The password is: "I know how to start assign-1"
%}
A       a
B       b
AB      ab
%%
{A}     {printf("This is character a.\n");}
{B}     {printf("This is character b.\n");}
{AB}    {printf("ab – The principle of longest
        substring!\n");}
%%
int Main()
{
        yylex();
        return 0;
}
```

Flex input file

Example sample1.flex

- What happens if the input is "**Hello**"?
- How to catch **\n**, only for empty lines?

```
%{/* This is a simple scanner. If you say Hello,
it will say Hi! If you say How are you? it will
say I'm fine. If you say anything else it will
say What? */
#include <stdio.h>
%}
GREATING1       "Hello“
GREATING2       "How are you?“
newline         \n
ANY             .*
%%
{GREATING1}     {printf("yylex()::Hi!\n");}
{GREATING2}     {printf("yylex()::I'm fine.\n");}
{newline}       {printf("newline\n");}
{ANY}           {printf("yylex()::What?\n");}
%%
int main(){
        yylex();
        return 0;
}
```

Flex input file

Example sample2.flex

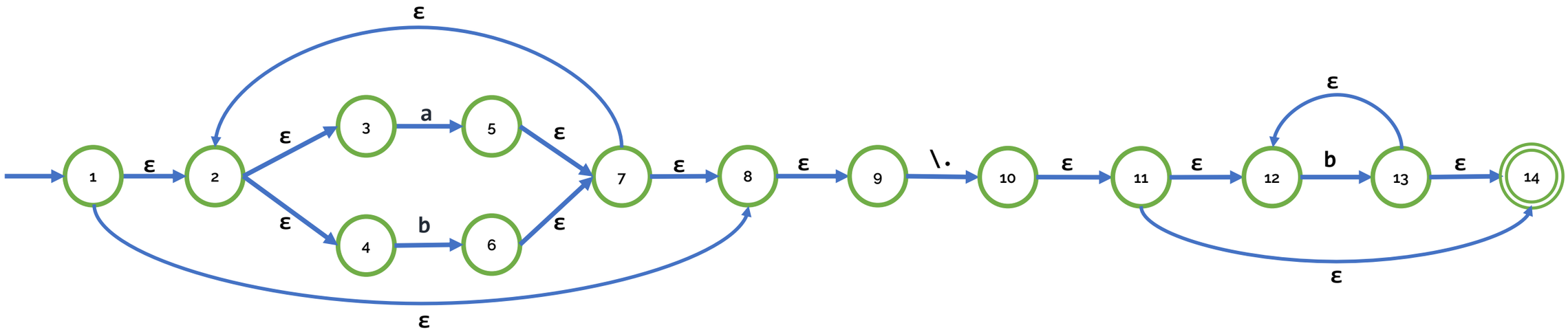
- Which part is related to the principle of longest substring?

```
%{/* This is a simple scanner for understating
how to catch special symbols like \n. It also
demonstrate how to use The Principle of Longest
Substring.*/
#include <stdio.h>
%}
newline          \n
ANY              .*\n
%%
{newline}        {printf("newline\n");}
{ANY}            {printf("yylex()::What?\n");}
%%
int main(){
        yylex();
        return 0;
}
```

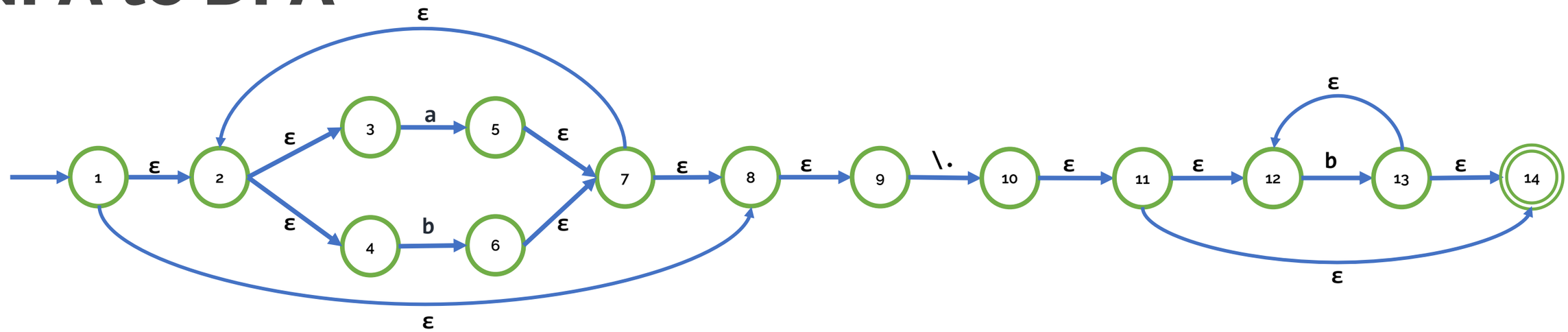
Flex input file

Example of Regular Expression to NFA

Here is the NFA for `(a|b)*\.b*`



NFA to DFA



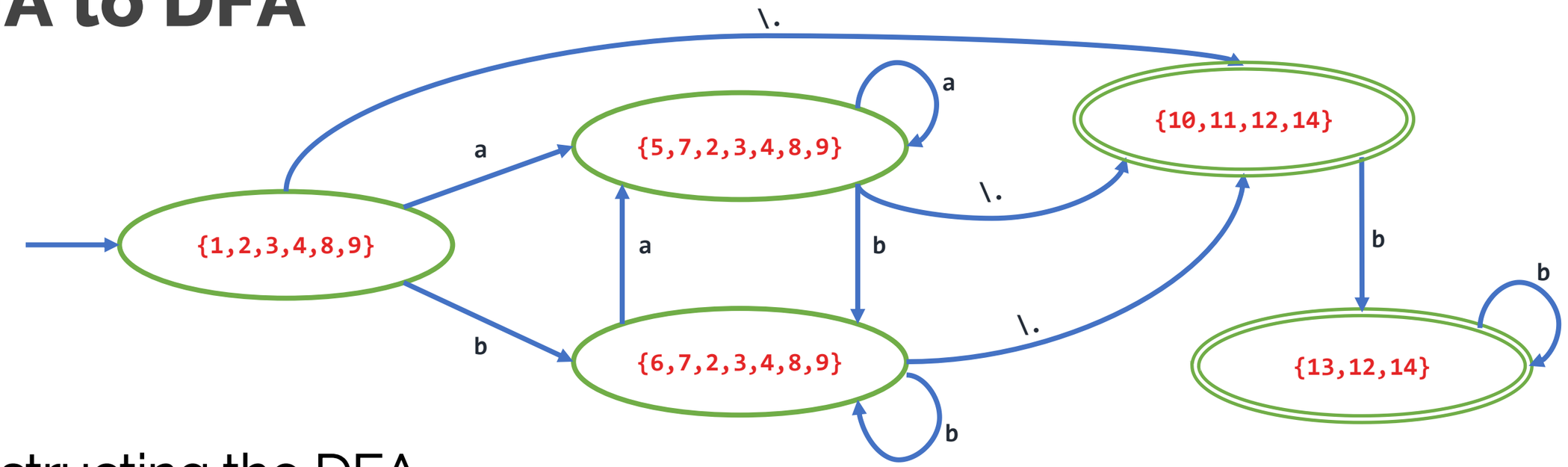
ε-closure of set of states:

$\overline{\{1\}} = \{1,2,3,4,8,9\}$ $\overline{\{2\}} = \{2,3,4\}$ $\overline{\{3\}} = \{3\}$ $\overline{\{4\}} = \{4\}$ $\overline{\{5\}} = \{5,7,2,3,4,8,9\}$

$\overline{\{6\}} = \{6,7,2,3,4,8,9\}$ $\overline{\{7\}} = \{7,2,3,4,8,9\}$ $\overline{\{8\}} = \{8,9\}$ $\overline{\{9\}} = \{9\}$ $\overline{\{10\}} = \{10,11,12,14\}$

$\overline{\{11\}} = \{11,12,14\}$ $\overline{\{12\}} = \{12\}$ $\overline{\{13\}} = \{13,12,14\}$ $\overline{\{14\}} = \{14\}$

NFA to DFA



Constructing the DFA:

$\overline{\{1\}} = \overline{\{1,2,3,4,8,9\}_a} = \overline{\{5\}} = \{5,7,2,3,4,8,9\}$ $\overline{\{1,2,3,4,8,9\}_b} = \overline{\{6\}} = \{6,7,2,3,4,8,9\}$ $\overline{\{1,2,3,4,8,9\}_{\backslash.}} = \overline{\{10\}} = \{10,11,12,14\}$

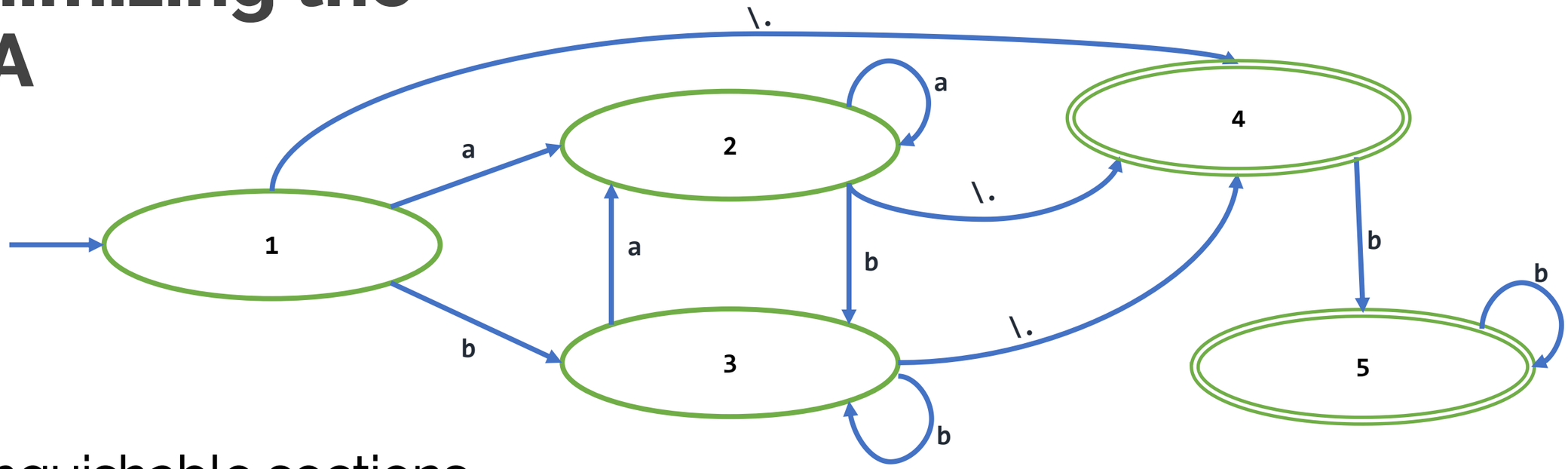
$\overline{\{5,7,2,3,4,8,9\}_a} = \overline{\{5\}}$ $\overline{\{5,7,2,3,4,8,9\}_b} = \overline{\{6\}}$ $\overline{\{5,7,2,3,4,8,9\}_{\backslash.}} = \overline{\{10\}}$

$\overline{\{6,7,2,3,4,8,9\}_a} = \overline{\{5\}}$ $\overline{\{6,7,2,3,4,8,9\}_b} = \overline{\{6\}}$ $\overline{\{6,7,2,3,4,8,9\}_{\backslash.}} = \overline{\{10\}}$

$\overline{\{10,11,12,14\}_a} = \emptyset$ $\overline{\{10,11,12,14\}_b} = \overline{\{13\}} = \{13,12,14\}$ $\overline{\{10,11,12,14\}_{\backslash.}} = \emptyset$

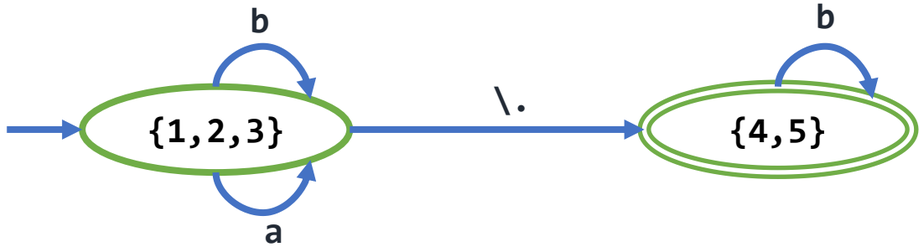
$\overline{\{13,12,14\}_a} = \emptyset$ $\overline{\{13,12,14\}_b} = \overline{\{13\}}$ $\overline{\{13,12,14\}_{\backslash.}} = \emptyset$

Minimizing the DFA



Distinguishable sections:

Accepting and non-accepting: {4,5} {1,2,3}
Character **a**: {4,5} {1,2,3}
Character **b**: {4,5} {1,2,3}
Character **.**: {4,5} {1,2,3}



Final Minimized DFA

`(a|b)*\.b*`