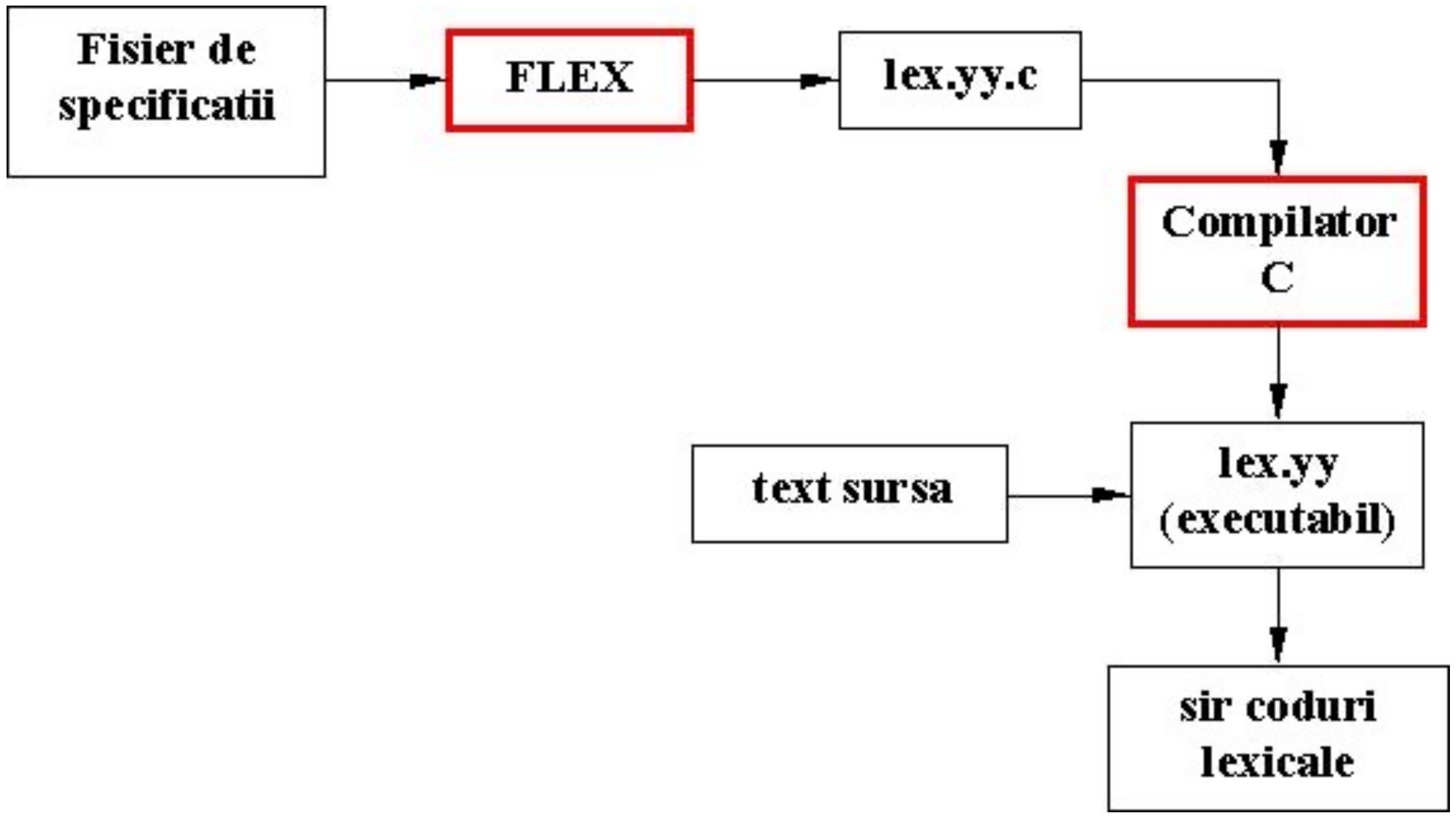
Course 3

Back to compiler construction

Scanning & Parsing Tools

- Scanning => lex
- Parsing => yacc //later

Lex – Unix utilitary (flex – Windows version)



INPUT FILE FORMAT

- The file containing the specification is a text file, that can have any name. Due to historic reasons we recommend the extension **.lxi**.
- Consists of 3 sections separated by a line containing %%:

```
definitions
%%
rules
%%
user code
```

Example 1:

```
%%

username printf( "%s", getlogin() );
```

specifies a scanner that, when finding the string "`username`", will replace it with the user login name

Definition Section:

- - declarations of simple *name definitions* (used to simplify the scanner specification), of the form

  `name definition`

- where:
- **name** is a word formed by one or more letters, digits, '_' or '-', with the remark that the first character MUST be letter or '_' and must be written on the FIRST POSITION OF THE LINE.
- **definition** is a regular expression and is starting with the first nonblank character after name until the end of line.
- declarations of *start conditions*.

Rules Section

- to associate semantic actions with regular expressions. It may also contain user defined C code, in the following way:

pattern action

where:

- **pattern** is a regular expression, whose first character MUST BE ON THE FIRST POSITION OF THE LINE; see RegExp file

- **action** is a sequence of one or more C statements that MUST START ON THE SAME LINE WITH THE PATTERN. If there are more than one statements they will be nested between {}. In particular, the action can be a void statement.

User Defined Code Section:

- Is optional (if is missing, then the separator %% following the rules section can also miss). If it exists, then its containing user defined C code is copied without any change at the end of the file lex.yy.c.
- Normally, in the user defined code section, one may have:

- function *main()* containing call(s) to *yylex()*, if we want the scanner to work autonomously (for ex., to test it);

- other called functions from *yylex()* (for ex. *yywrap()* or functions called during actions); in this case, the user code from definitions section must contain: either prototypes, either ***#include*** directives of the headers containing the prototypes

Launching the execution:

lex [*option*] [*name_specification _file*]

where *name_specification _file is an input file (implicitly,* stdin)

$ lex spec.lxi

$ gcc lex.yy.c -o your_lex

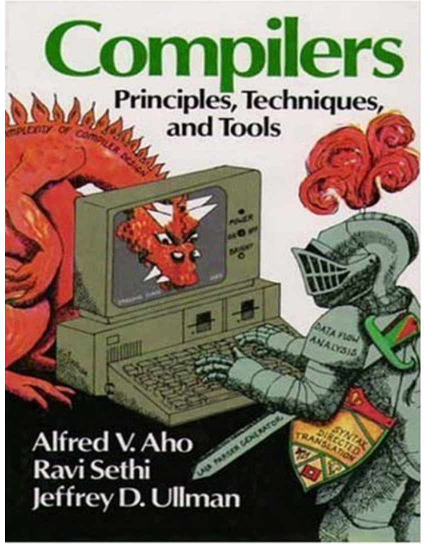
$ your_lex<input.txt

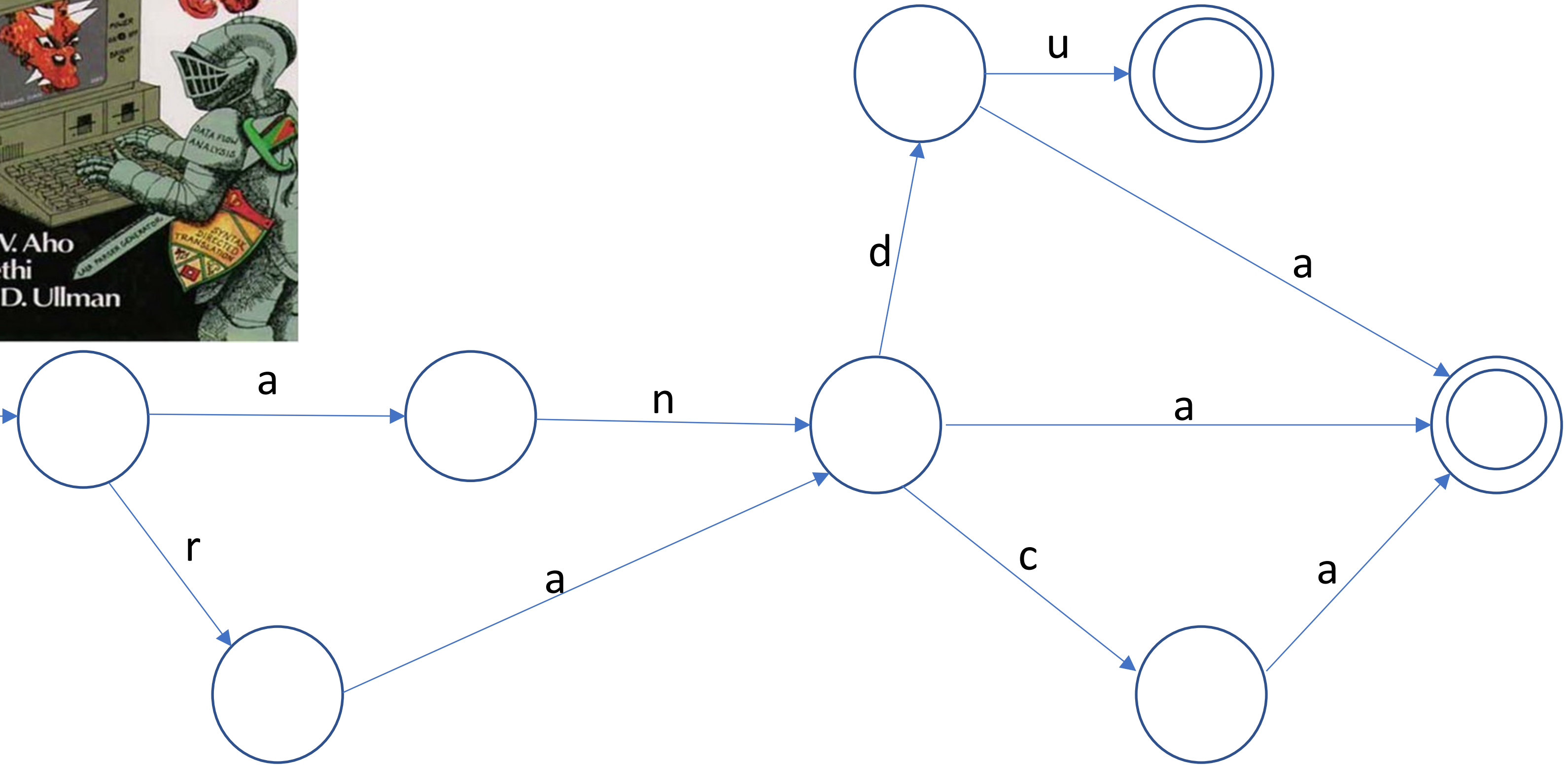
options: http://dinosaur.compilertools.net/flex/manpage.html

Example

Formal Languages

- basic notions-

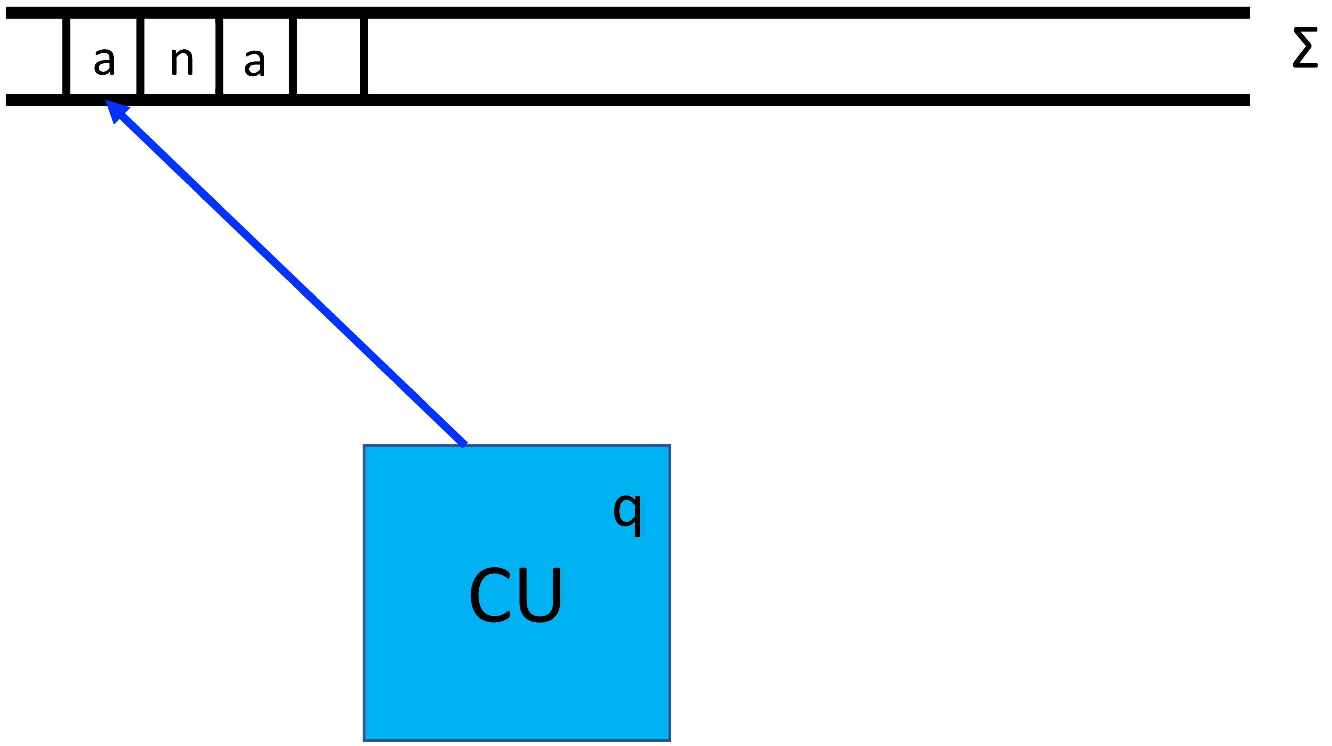




Problem: The door to the tower is closed by the Red Dragon, using a complicated machinery. Prince Charming has managed to steal the plans and is asking for your help. Can you help him determining all the person names that can unlock the door

Finite Automata (sing. Finite automaton; abrev. FA; transl = automat finit)

- Intuitive model



Definition: A ***finite automaton (FA)*** is a 5-tuple

$$M = (Q,\Sigma,\delta,q0,F)$$

where:

- Q - finite set of states ($|Q|<\infty$)
- Σ - finite alphabet ($|\Sigma|<\infty$)
- δ – transition function : $\delta:Q\times\Sigma\rightarrow P(Q)$
- q_0 – initial state $q_0 \in Q$
- $F\subseteq Q$ – set of final states

Remarks

1. $Q \cap \Sigma = \emptyset$
2. $\delta: Q \times \Sigma \rightarrow P(Q)$, $\varepsilon \in \Sigma^0$ - relation $\delta(q,\varepsilon)=p$ **NOT** allowed
3. If $|\delta(q,a)| \leq 1$ => deterministic finite automaton (DFA)
4. If $|\delta(q,a)| > 1$ (more than a state obtained as result) => nondeterministic finite automaton (NFA)

Property: For any NFA *M* there exists a DFA *M'* equivalent to *M*

Configuration $C=(q,x)$

where:

- q state
- x unread sequence from input: $x \in \Sigma^*$

Initial configuration : (q_0,w) , w - whole sequence

Final configuration: (q_f ,ε) , $q_f \in F$, ε –empty sequence

(corresponds to accept)

Relations between configurations

- $\vdash$ **move / transition** (simple, one step)
  $(q,ax) \vdash (p,x)$, $p \in \delta(q,a)$

- $\overset{k}{\vdash}$ **k move** = a sequence of k simple transitions) $C_0 \vdash C_1 \vdash \ldots \vdash C_k$
- $\overset{+}{\vdash}$ **+ move**
  $C \overset{+}{\vdash} C'$: $\exists$ $k>0$ such that $C \overset{k}{\vdash} C'$
- $\overset{*}{\vdash}$ *** move (star move)**
  $C \overset{*}{\vdash} C'$: $\exists$ $k \geq 0$ such that $C \overset{k}{\vdash} C'$

Definition : ***Language*** accepted by FA $M = (Q,\Sigma,\delta,q0,F)$ is:

$$L(M)=\{ w \in \Sigma^* \mid (q_0,w) \vdash^* (q_f ,\varepsilon) , q_f \in F \}$$

Remarks

1. 2 finite automata M_1 and M_2 are equivalent if and only if they accept the same language

$$L(M_1)=L(M_2)$$

1. $\varepsilon \in L(M) \Leftrightarrow q_0 \in F$ (initial state is final state)

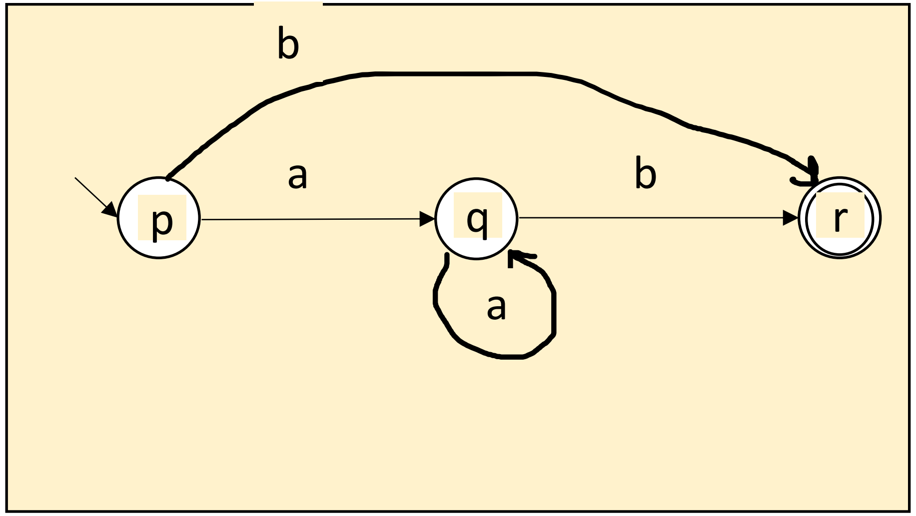
Representing FA

1. List of all elements
2. Table
3. Graphical representation

M=(Q,Σ,δ,p,F)
Q = {p,q,r}
Σ = {a,b}
δ(p,a) = q
δ(q,a)=q
δ(q,b)=r
δ(p,b)=r
F = {r}

M=(Q,Σ,δ,p,F)
F = {r}

	a	b
p	q	r
q	q	r
r	-	-



Example

M=(Q,Σ,δ,p,F)

Q = {p,q,r,s}

Σ = {0,1}

δ(p,1) = q

δ(q,0)=q

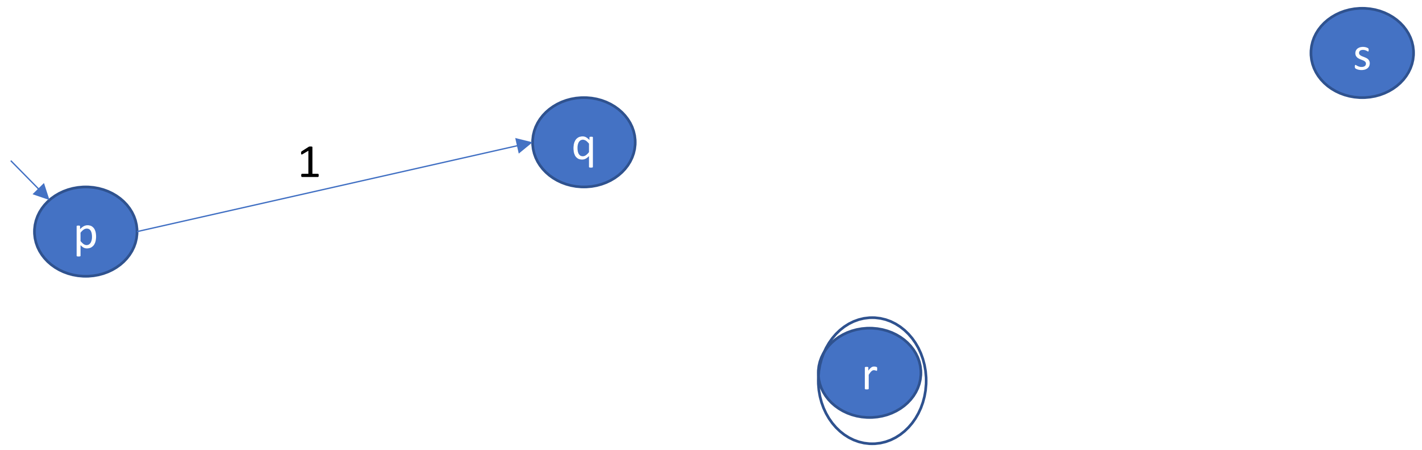
δ(q,1)=r

δ(p,0)=s

δ(s,1)=s

δ(s,0)=r

F = {r}



(p,101)|-(q,01)|-(q,1)|-(r, ε) accepted

(p,110) |-(q,10)|-(r,0) –not accepted

F = {p,r}

Regular languages

Why?

1. Search engine – succes of Google
2. Unix commands
3. Programming languages – new feature

Remember

- **Grammar**

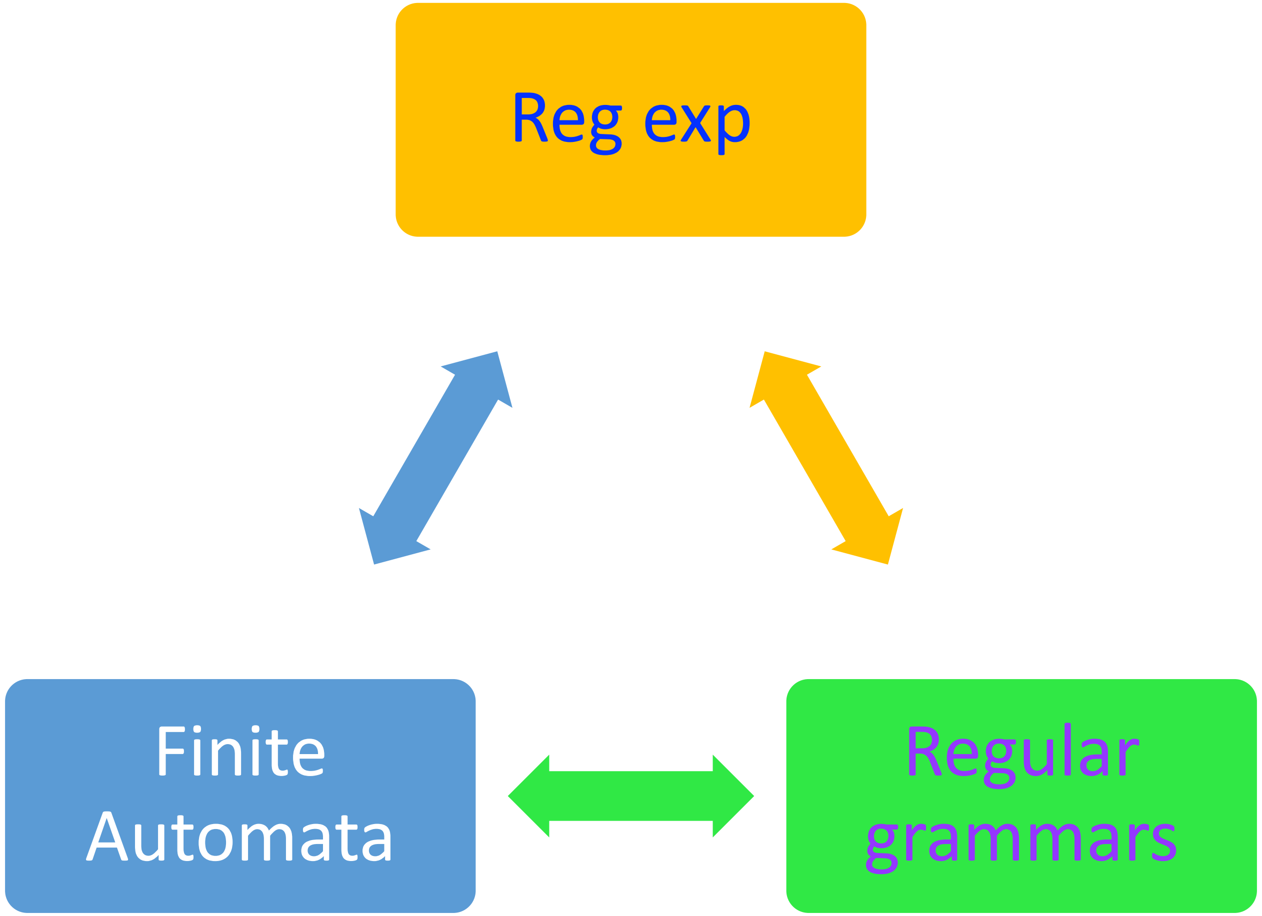
$G=(N,\Sigma,P,S)$

$L(G)=\{w\in\Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$

- Finite automaton

$M = (Q,\Sigma,\delta,q_0,F)$

$L(M)=\{ w \in \Sigma^* \mid (q_0,w) \vdash (q_f ,\varepsilon) , q_f \in F \}$


Reg exp
Finite Automata
Regular grammars