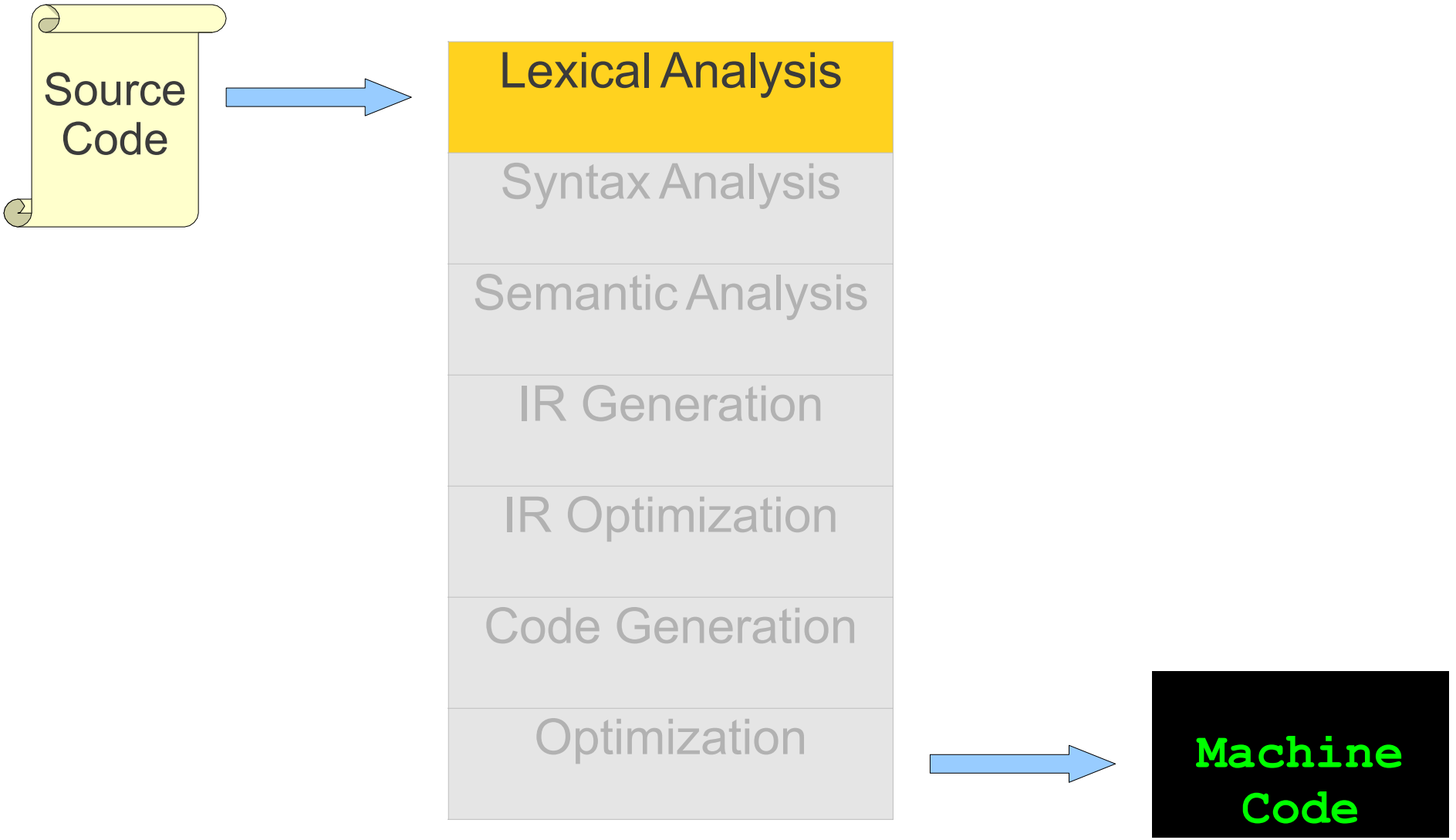
بسم الله الرحمن الرحيم

Lexical Analysis (2)

Review

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Scanner

- function getNextToken() returns
  - type of next token
  - attribute of the token (e.g. integer constant, comparison type, …)

Scanning a Source File

The piece of the original program from which we made the token is called a **lexeme**.

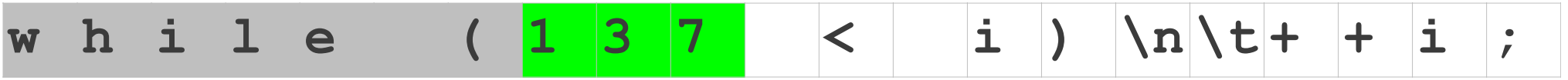
This is called a **token**. You can think of it as an enumerated type representing what logical entity we read out of the source code.

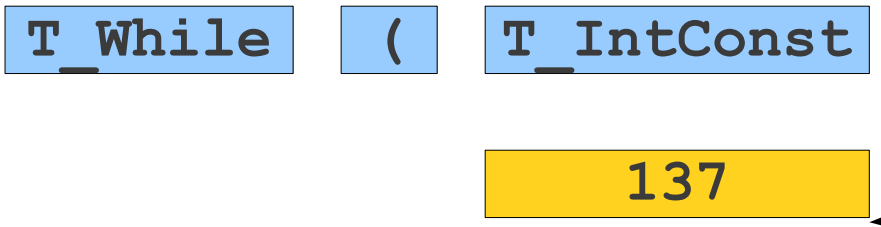
Scanning a Source File

T_While

Sometimes we will discard a lexeme rather than storing it for later use.
Here, we ignore whitespace, since it has no bearing on the meaning of the program.

Scanning a Source File





Some tokens can have **attributes** that store extra information about the token. Here we store which integer is represented.

Scanning is Hard

- FORTRAN: Whitespace is irrelevant

```
DO 5 I = 1,25
DO5I   = 1.25
```

- Can be difficult to tell when to partition input.

Thanks to Prof. Alex Aiken

Scanning is Hard

- C++: Nested template declarations

```
vector<vector<int>> myVector
```

Thanks to Prof. Alex Aiken

Scanning is Hard

- C++: Nested template declarations

```
vector < vector < int >> myVector
```

Thanks to Prof. Alex Aiken

Scanning is Hard

- C++: Nested template declarations

```
(vector < (vector < (int >> myVector)))
```

Thanks to Prof. Alex Aiken

Scanning is Hard

- C++: Nested template declarations

```
(vector < (vector < (int >> myVector)))
```

- Again, can be difficult to determine where to split.

Thanks to Prof. Alex Aiken

Scanning is Hard

- PL/1: Keywords can be used as identifiers.

Thanks to Prof. Alex Aiken

Scanning is Hard

- PL/1: Keywords can be used as identifiers.

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```

Thanks to Prof. Alex Aiken

Scanning is Hard

- PL/1: Keywords can be used as identifiers.

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```

Thanks to Prof. Alex Aiken

Scanning is Hard

- PL/1: Keywords can be used as identifiers.

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```

- Can be difficult to determine how to label lexemes.

Thanks to Prof. Alex Aiken

Challenges in Scanning

-
-
-

Challenges in Scanning

- How do we determine which lexemes are associated with each token?
-
-

Challenges in Scanning

- How do we determine which lexemes are associated with each token?
- When there are multiple ways we could scan the input, how do we know which one to pick?
-

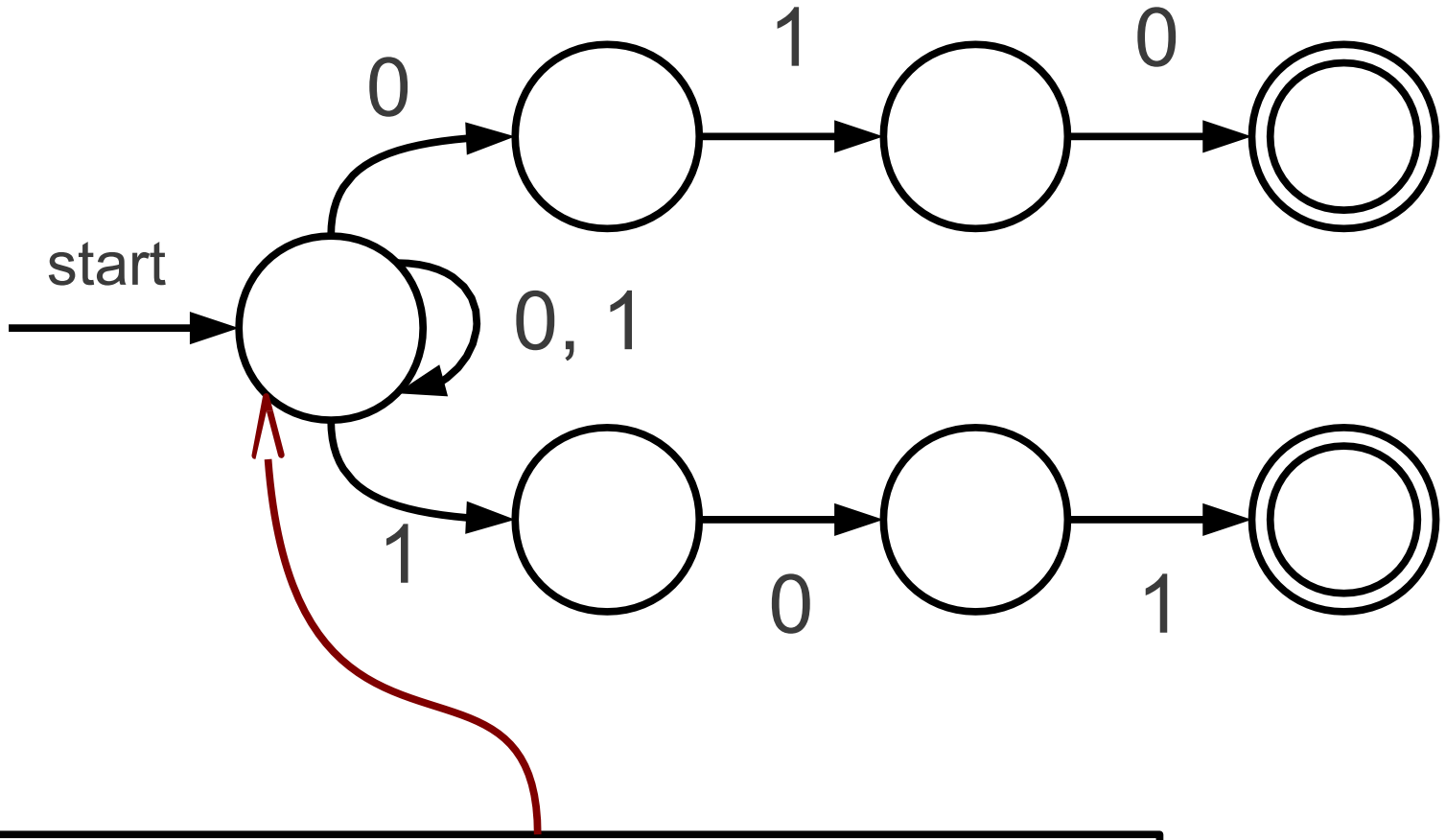
Challenges in Scanning

- How do we determine which lexemes are associated with each token?
- When there are multiple ways we could scan the input, how do we know which one to pick?
- How do we address these concerns efficiently?

Idea:

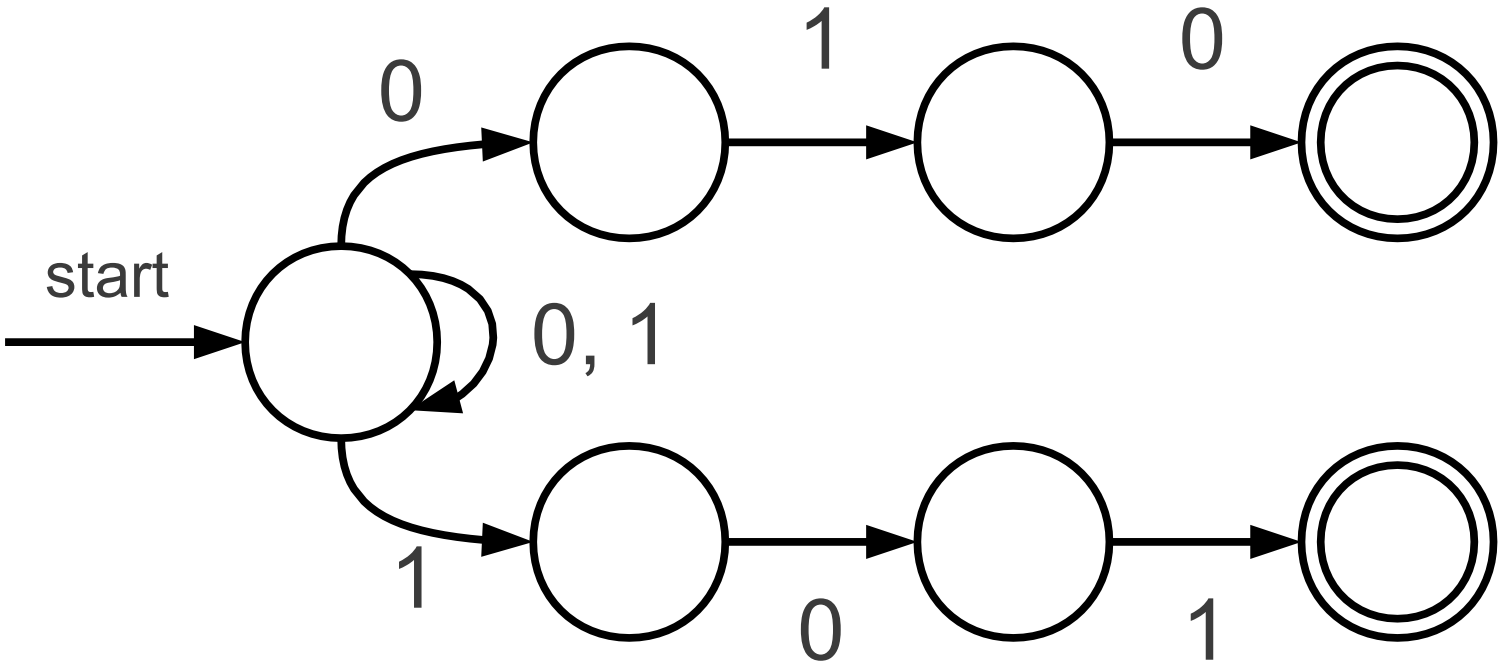
- To each token, assign a regular language
  - = regular expression

A More Complex Automaton



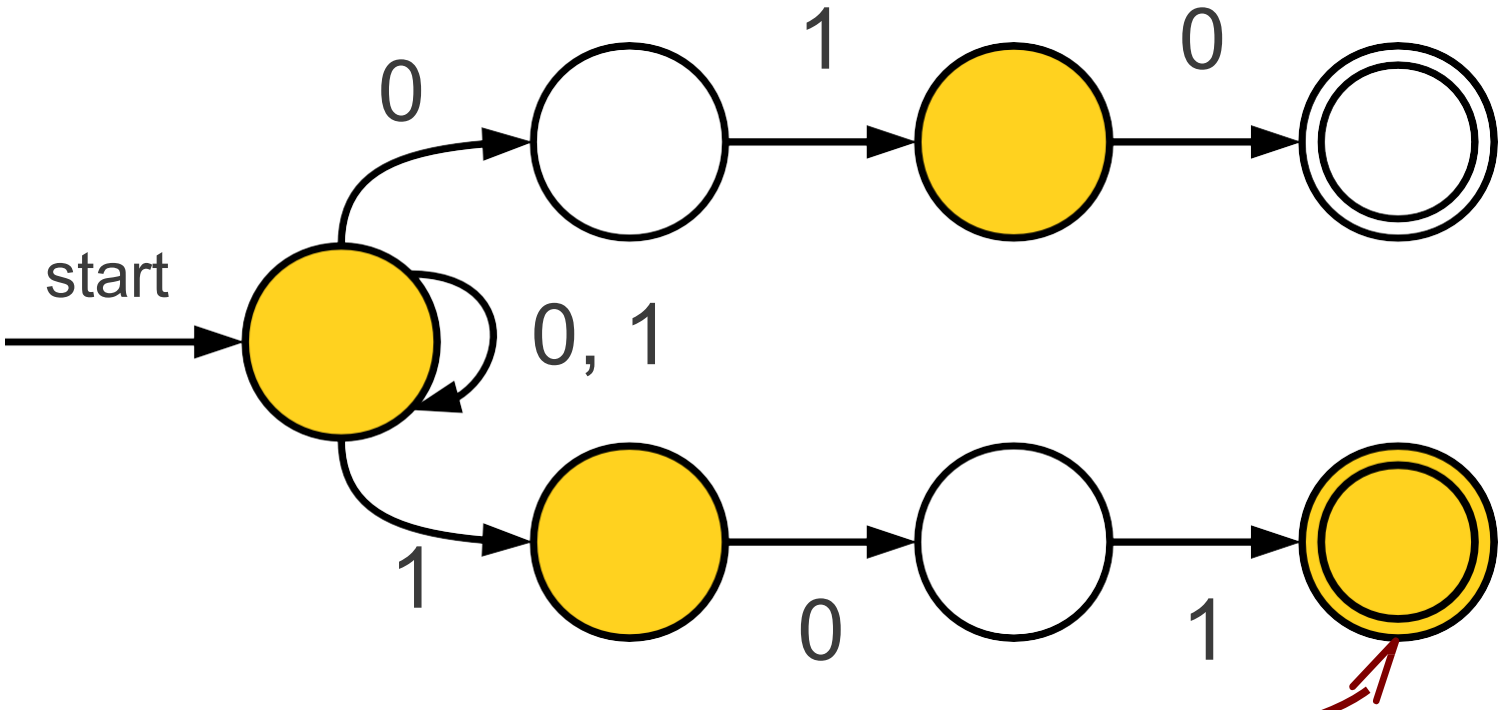
Notice that there are multiple transitions defined here on 0 and 1. If we read a 0 or 1 here, we follow *both* transitions and enter multiple states.

A More Complex Automaton



0	1	1	1	0	1

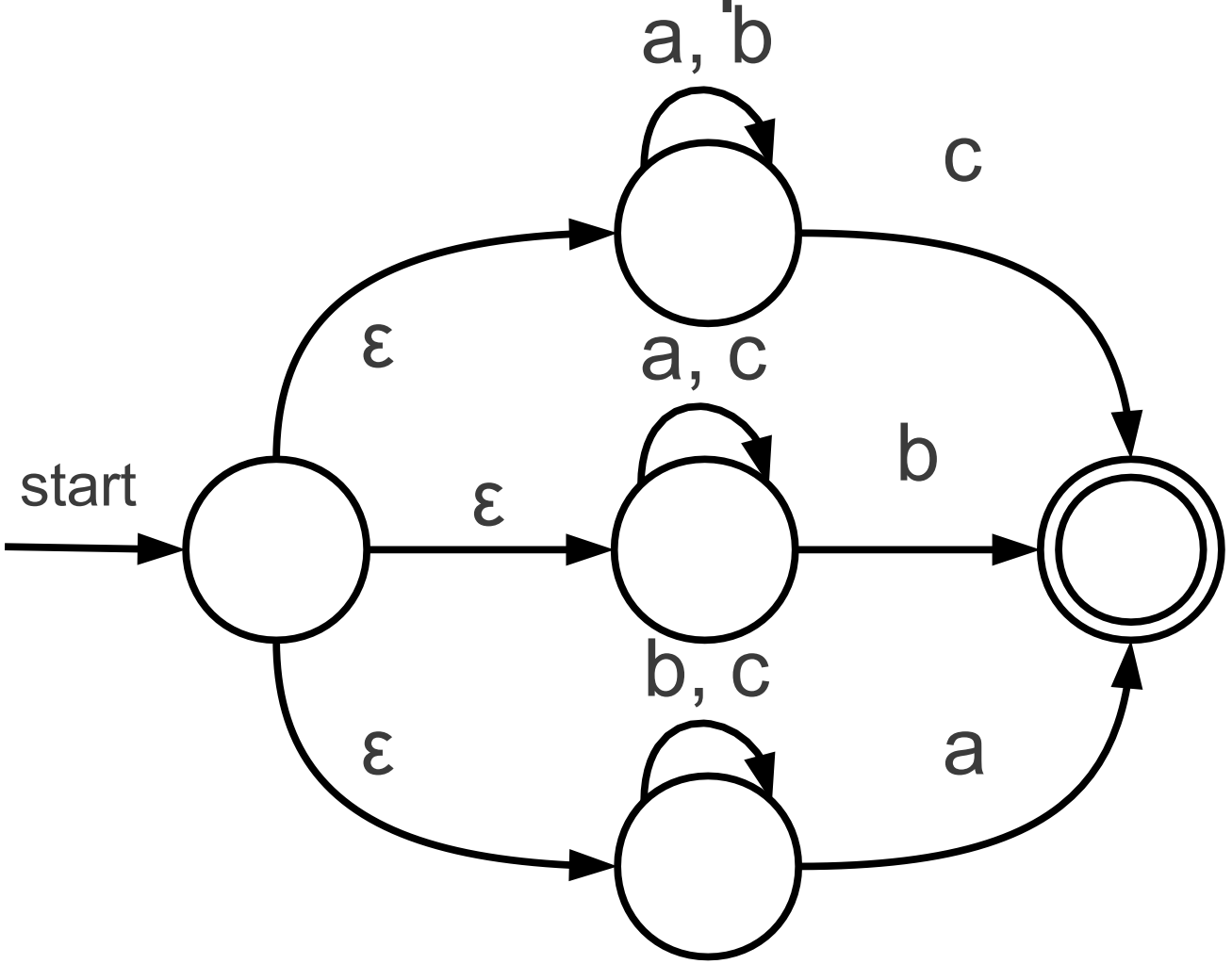
A More Complex Automaton



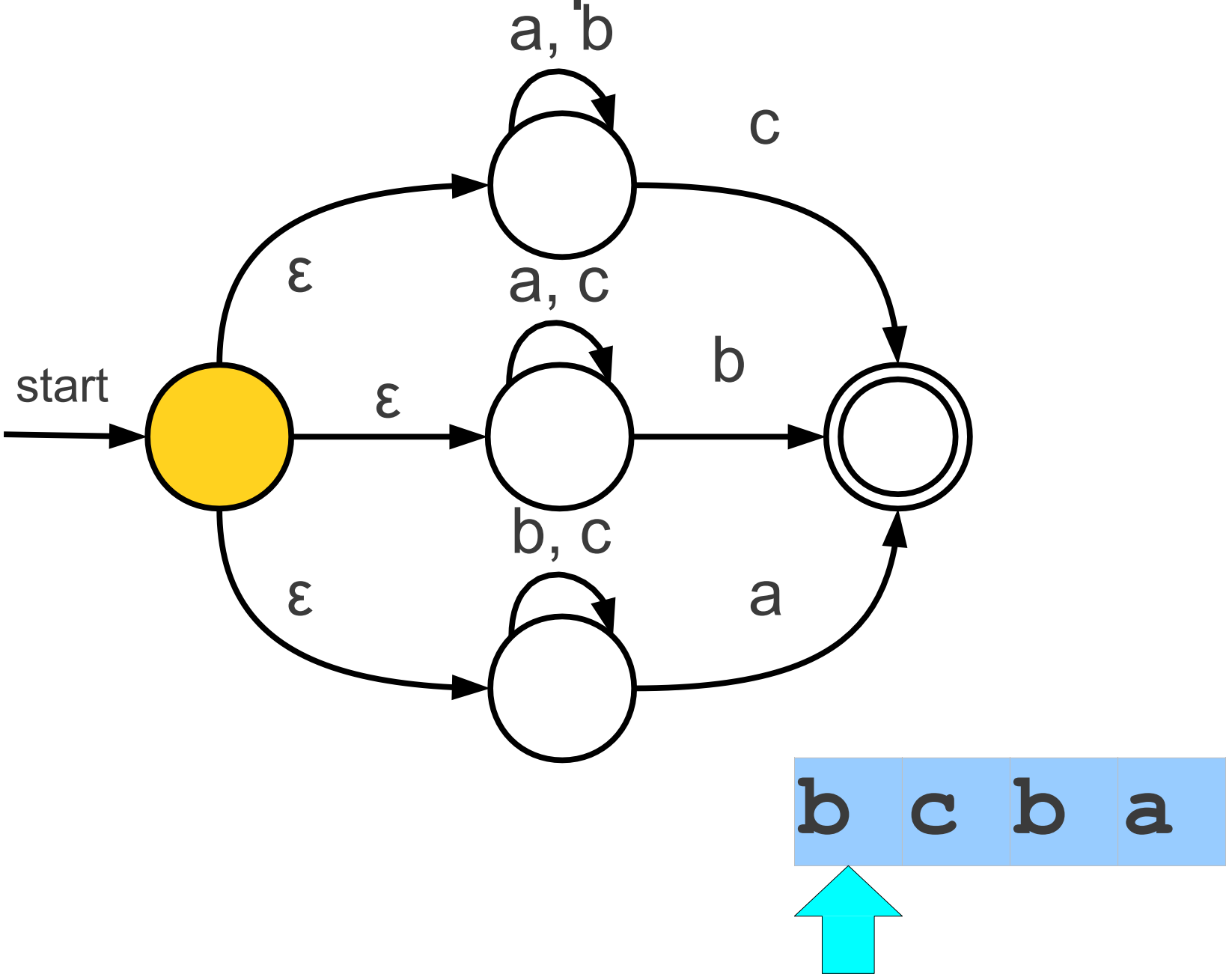
0	1	1	1	0	1

Sincewe are in at least one accepting state, the automaton accepts.

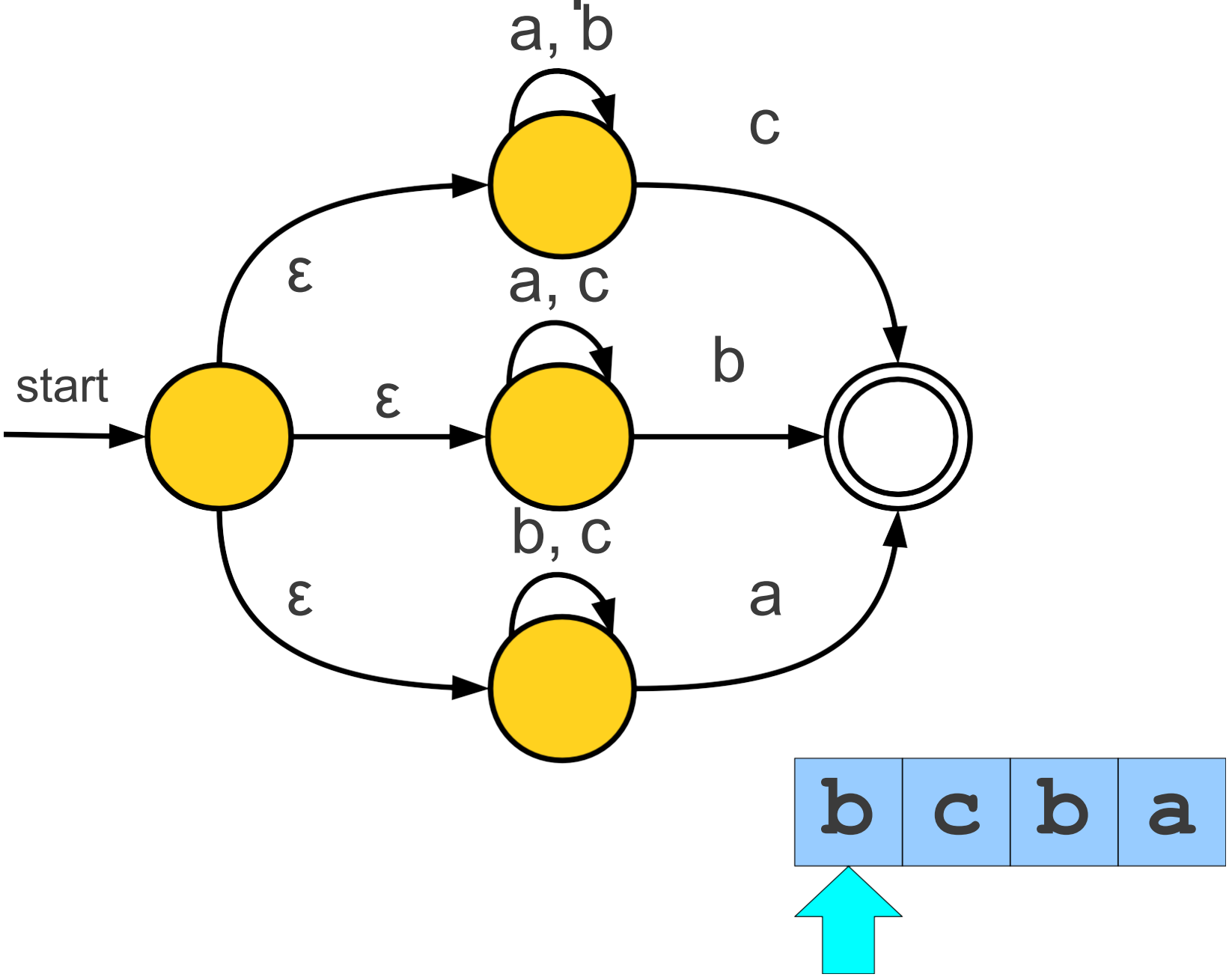
An Even More Complex Automaton



An Even More Complex Automaton

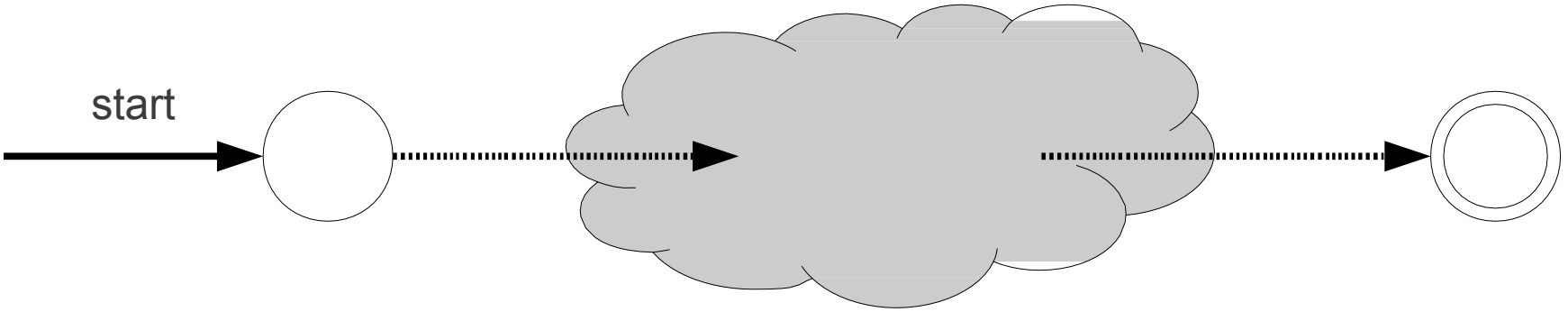


An Even More Complex Automaton

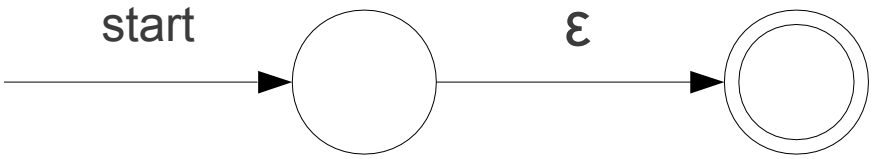


From Regular Expressions to NFAs

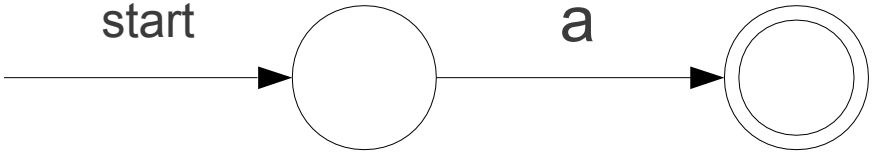
- There is a (beautiful!) procedure from converting a regular expression to an NFA.
- Associate each regular expression with an NFA with the following properties:
  - There is exactly one accepting state.
  - There are no transitions out of the accepting state. There are no transitions into the starting state.
- These restrictions are stronger than necessary, but make the construction easier.



Base Cases



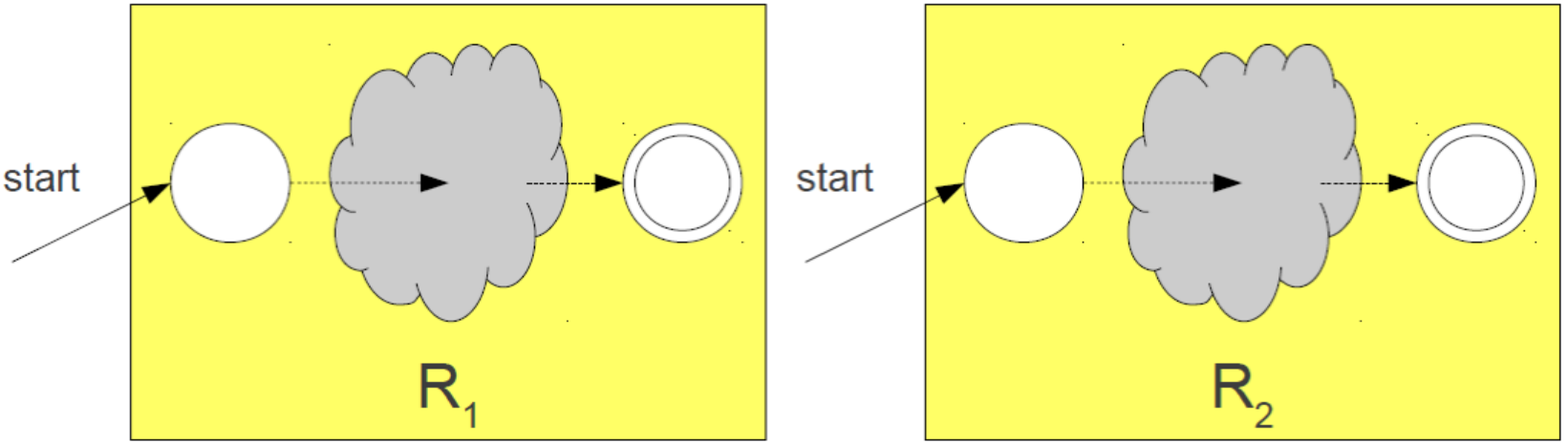
Automaton for ε



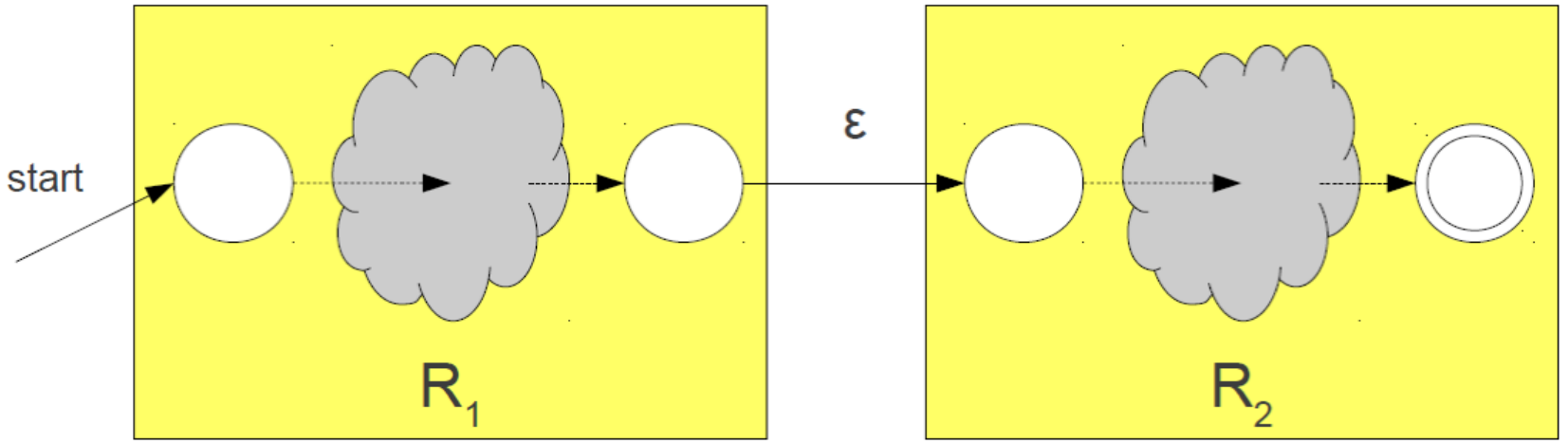
Automaton for single character **`a`**

Construction for R_1R_2

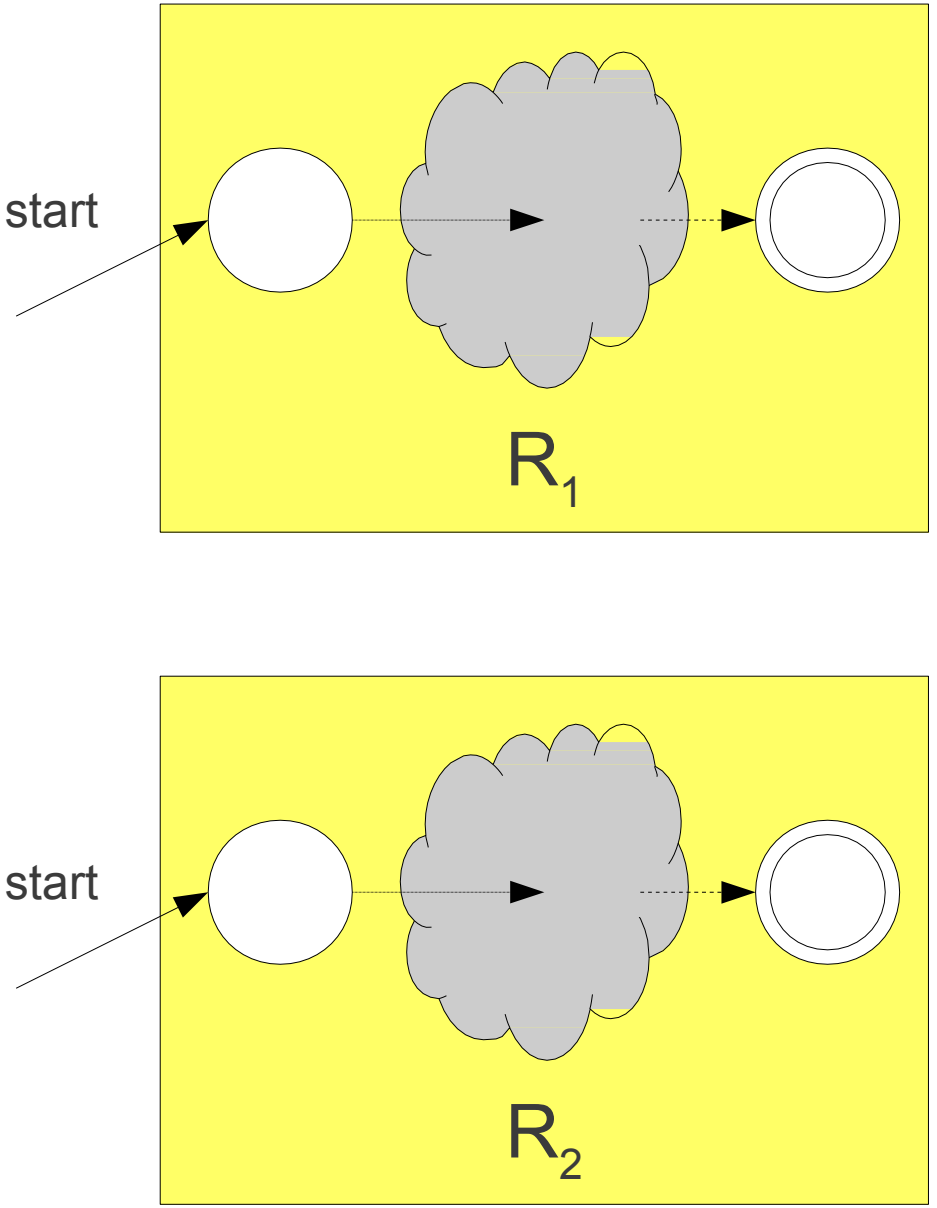
Construction for R_1R_2



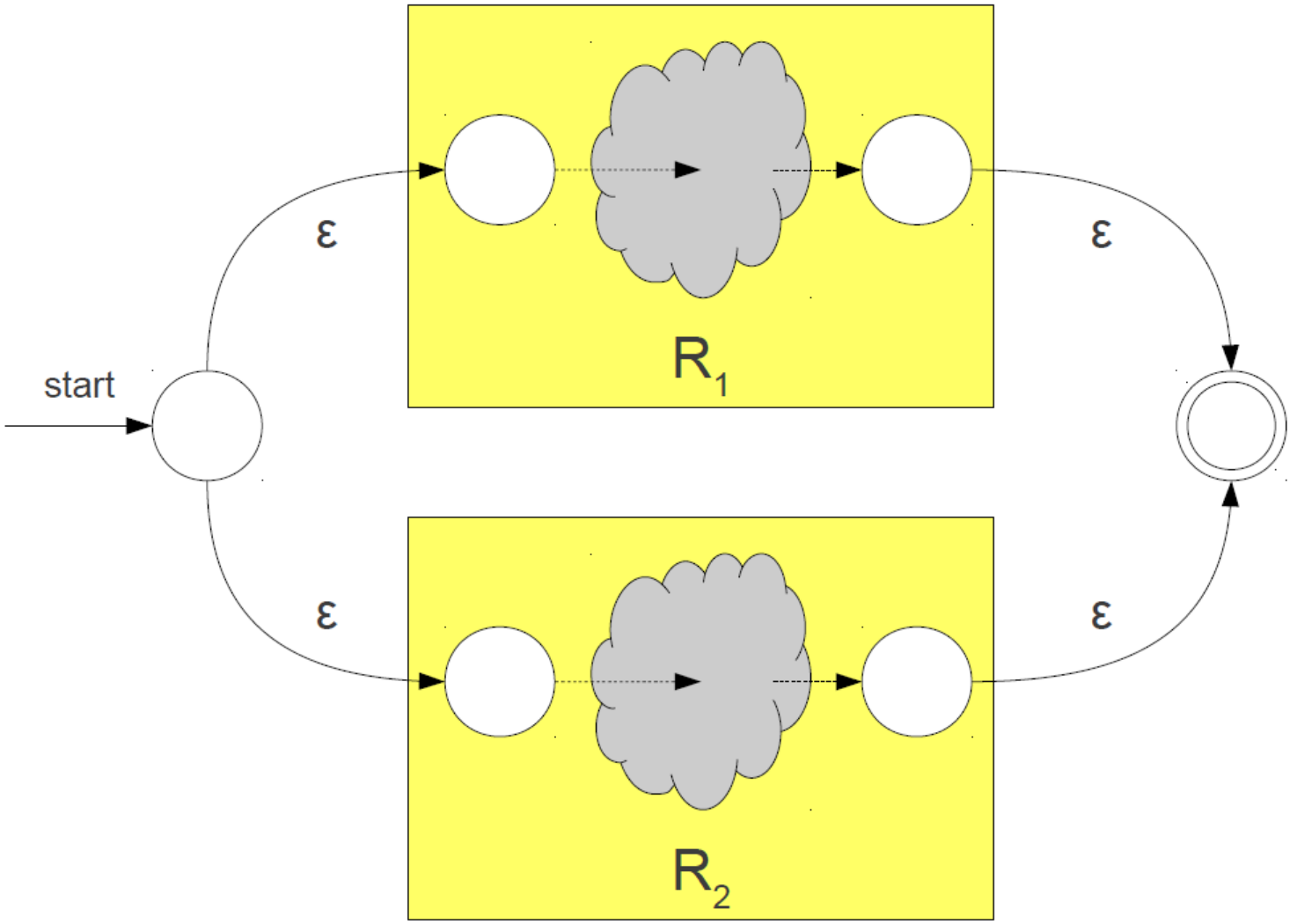
Construction for R_1R_2



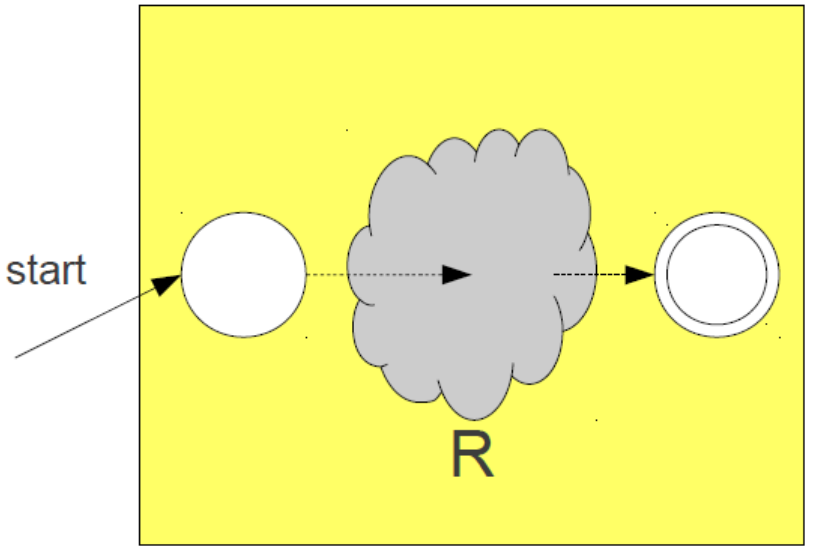
Construction for $R_1 \mid R_2$



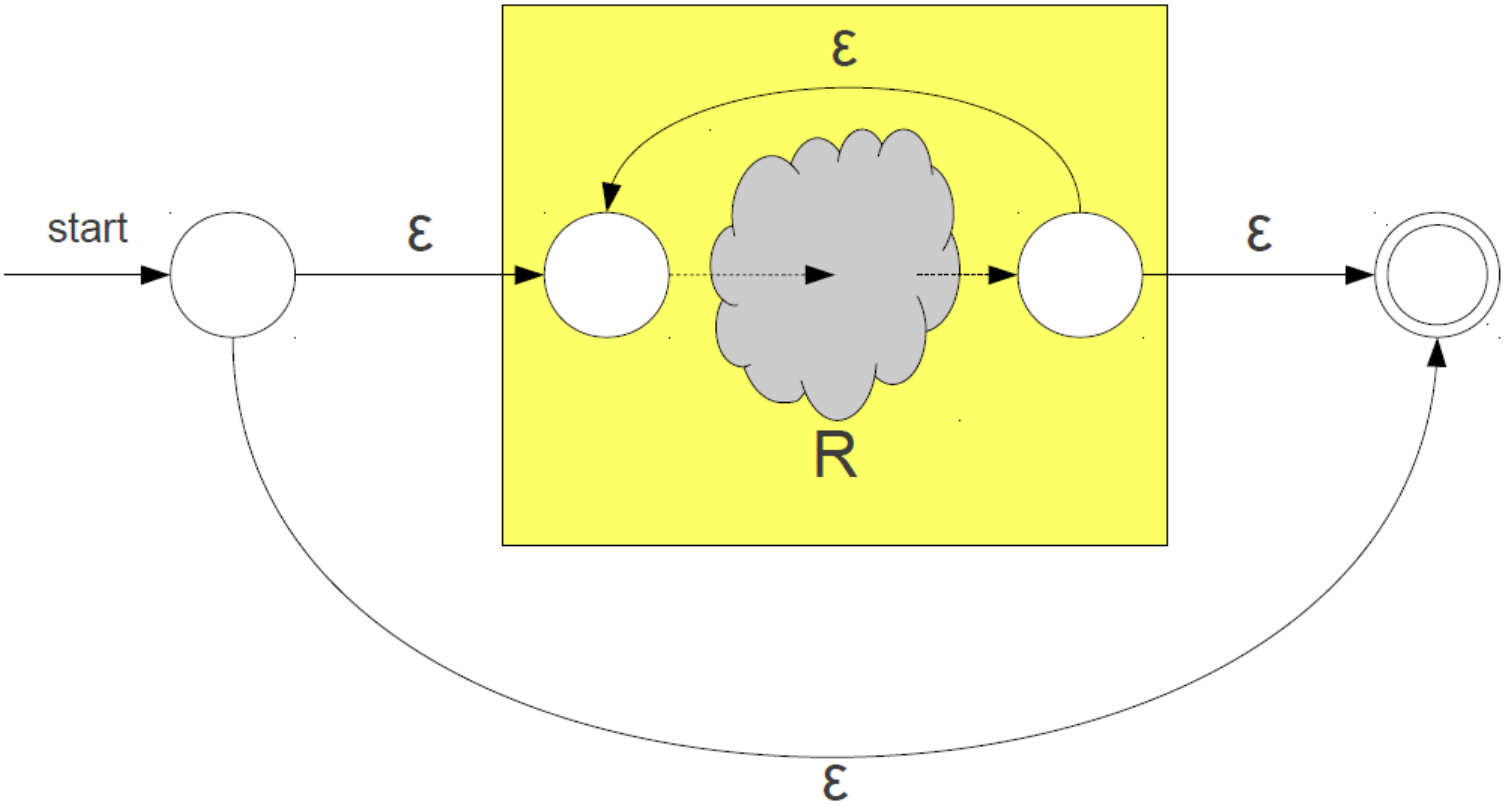
Construction for R_1 | R_2



Construction for R*



Construction for R*



Continue …

Overall Result

McNaughton–Yamada–Thompson Algorithm

- Any regular expression of length *n* can be converted into an NFA with O(*n)* states.
- Can determine whether a string of length *m* matches a regular expression of length *n* in time O(mn^2).
- We'll see how to make this O(*m*) later (this is independent of the complexity of the regular expression!)

Challenges in Scanning

- How do we determine which lexemes are associated with each token?
- When there are multiple ways we could scan the input, how do we know which one to pick?
- How do we address these concerns efficiently?

Lexing Ambiguities

```
T_For           for
T_Identifier    [A-Za-z_][A-Za-z0-9_]*
```

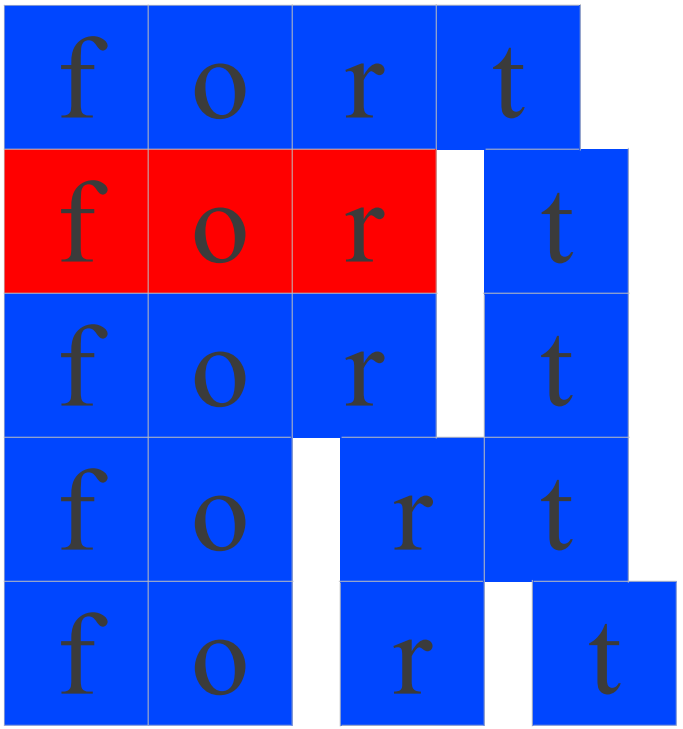
Lexing Ambiguities

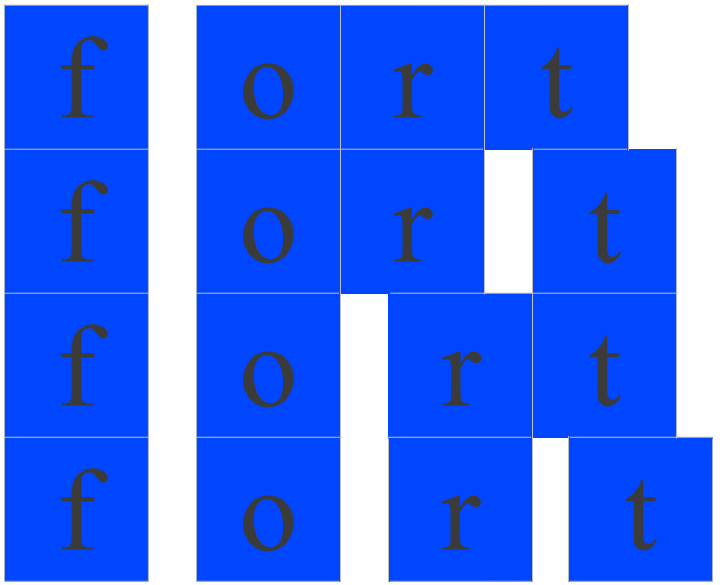
```
T_For          for
T_Identifier   [A-Za-z_][A-Za-z0-9_]*
```

f	o	r	t

Lexing Ambiguities

```
T_For                for
T_Identifier         [A-Za-z_][A-Za-z0-9_]*
```



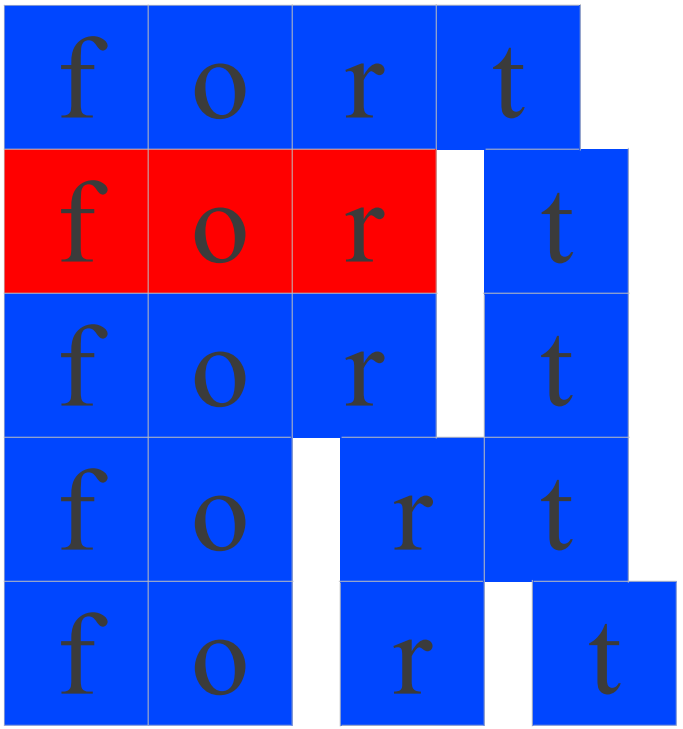


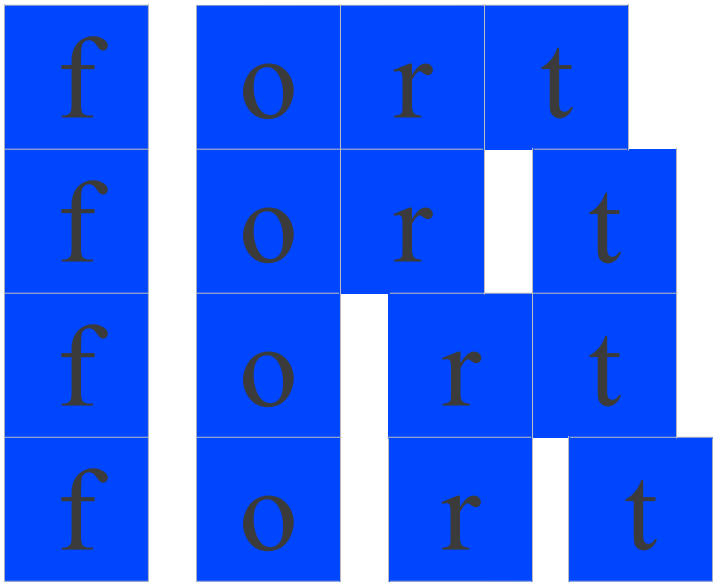
Conflict Resolution

- Assume all tokens are specified as regular expressions.
- Algorithm: **Left-to-right scan**.
- Tiebreaking rule one: **Maximal munch**.
  - Always match the longest possible prefix of the remaining text.

Lexing Ambiguities

```
T_For                for
T_Identifier         [A-Za-z_][A-Za-z0-9_]*
```





Lexing Ambiguities

```
T_For          for
T_Identifier   [A-Za-z_][A-Za-z0-9_]*
```

Implementing Maximal Munch

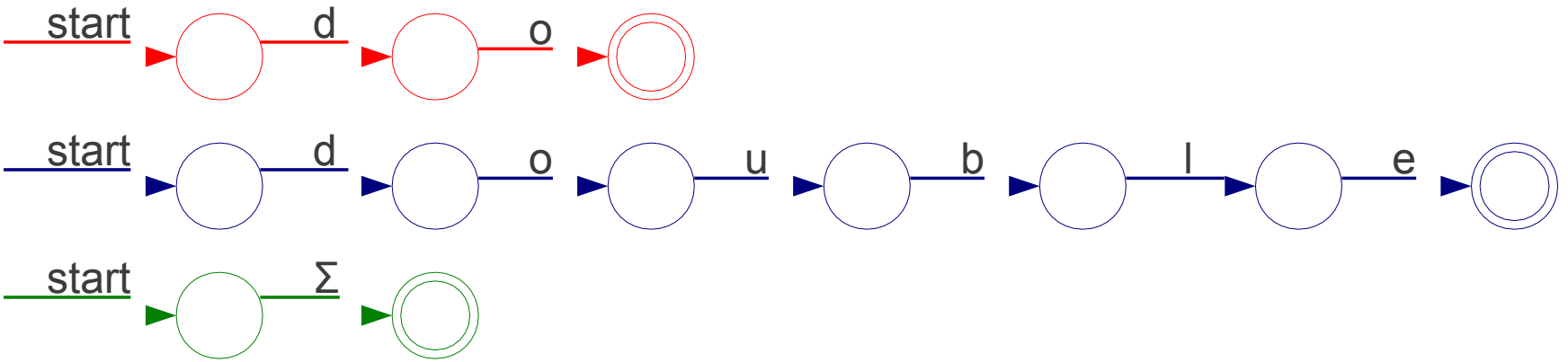
- Given a set of regular expressions, how can we use them to implement maximum match?
- Idea:
  - Convert expressions to NFAs.
  - Run all NFAs in parallel, keeping track of the last match.
  - When all automata get stuck, report the last match and restart the search at that point.

Implementing Maximal Munch

```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

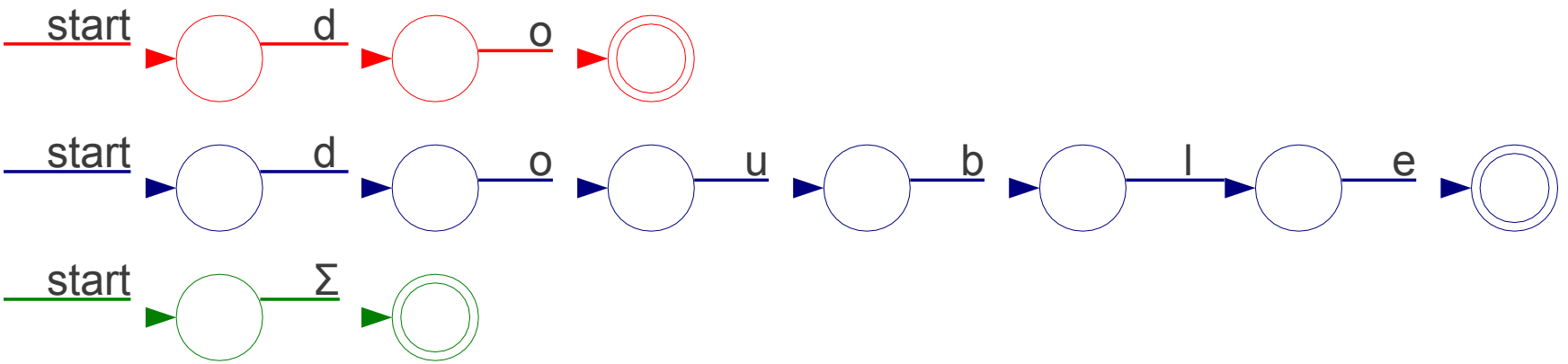
Implementing Maximal Munch

```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```



Implementing Maximal Munch

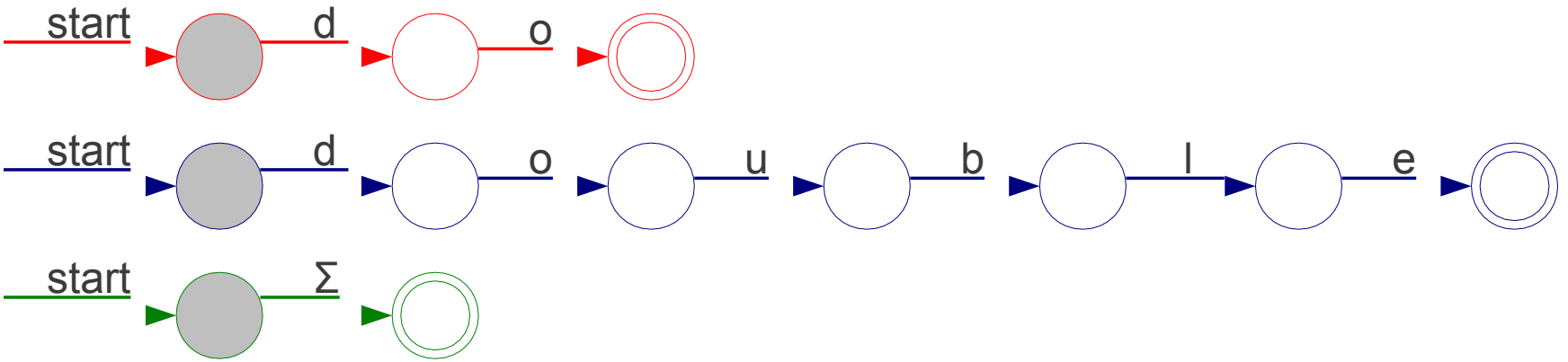
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```



D	O	U	B	D	O	U	B	L	E

Implementing Maximal Munch

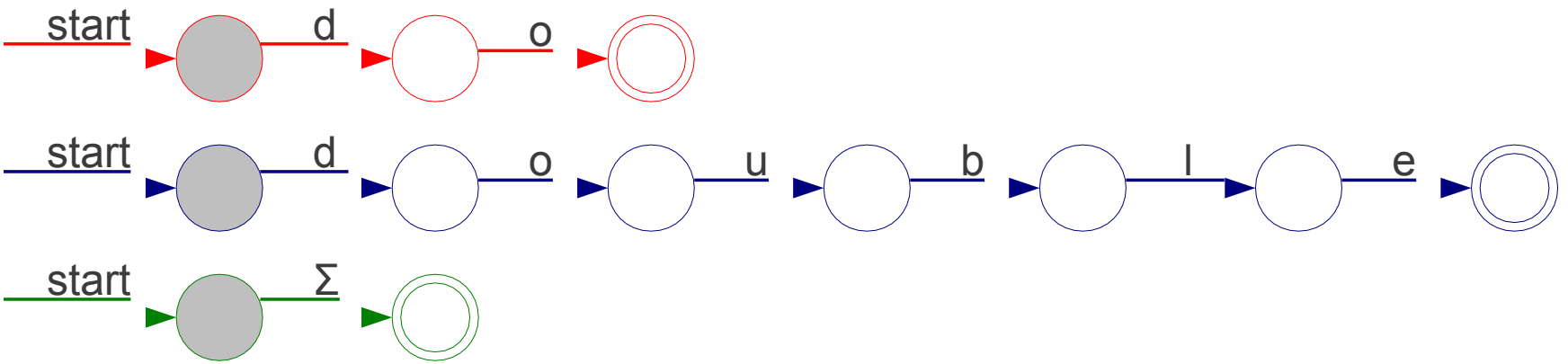
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```



D	O	U	B	D	O	U	B	L	E

Implementing Maximal Munch

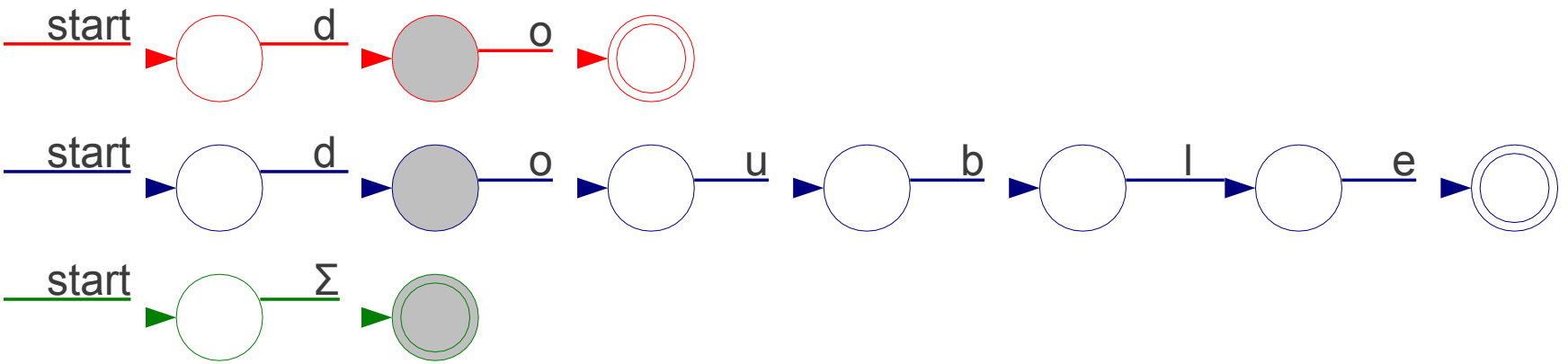
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```



D	O	U	B	D	O	U	B	L	E

Implementing Maximal Munch

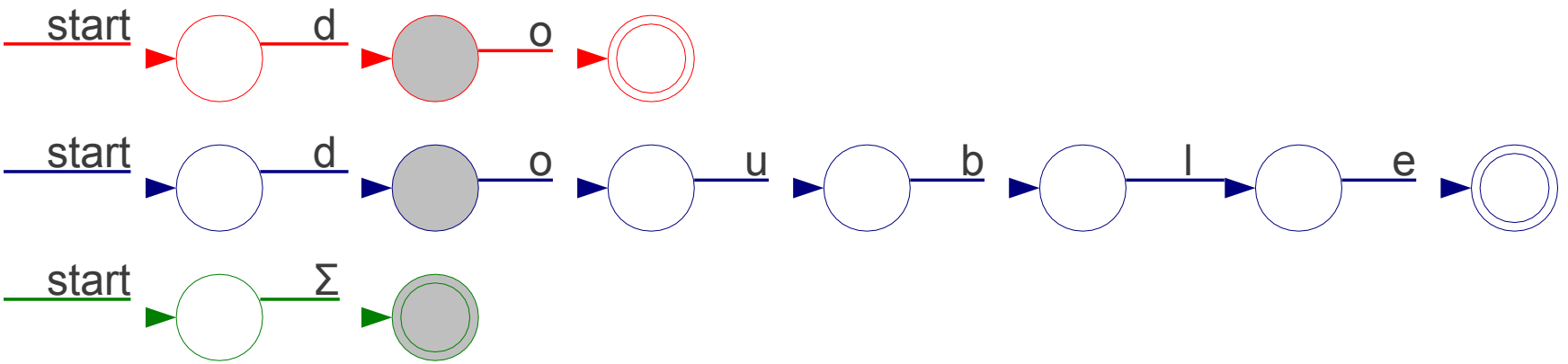
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

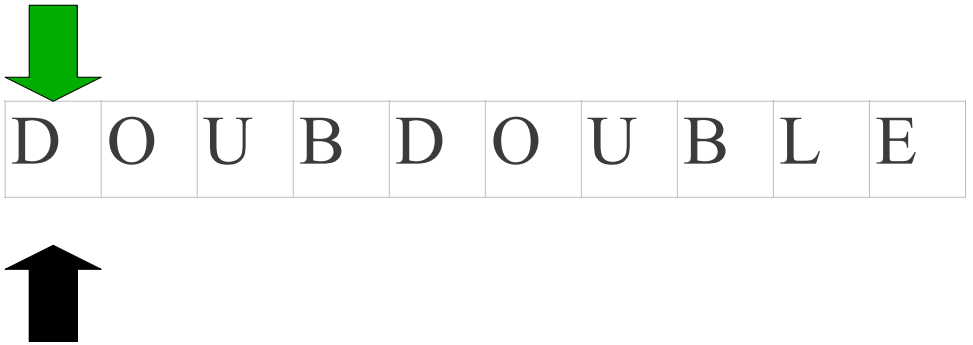


D	O	U	B	D	O	U	B	L	E

Implementing Maximal Munch

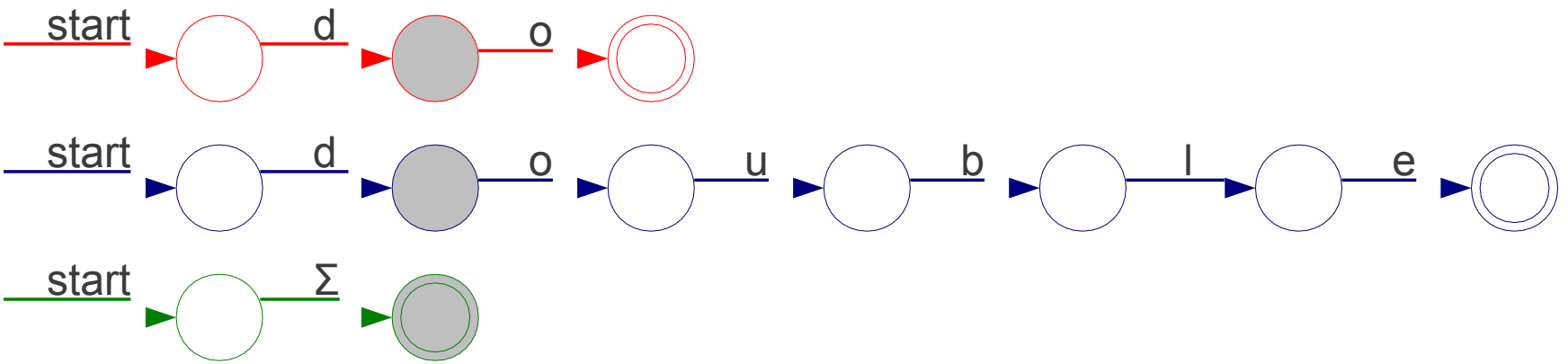
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

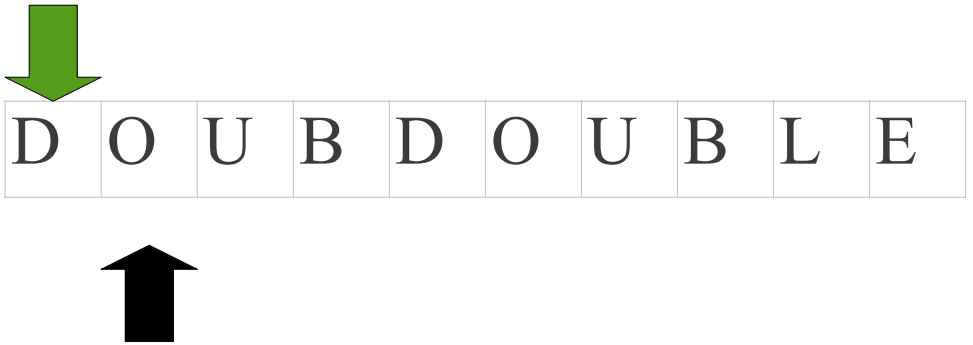




Implementing Maximal Munch

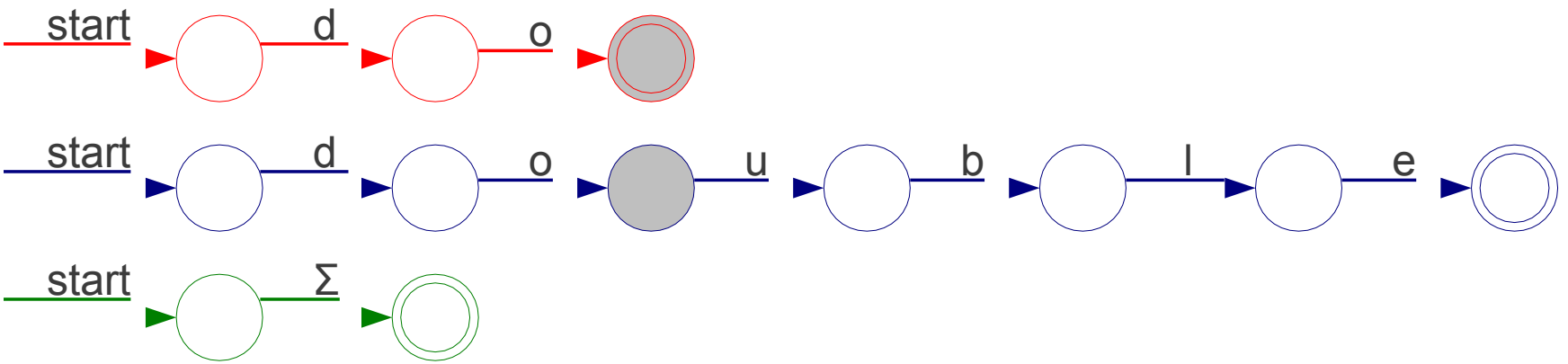
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

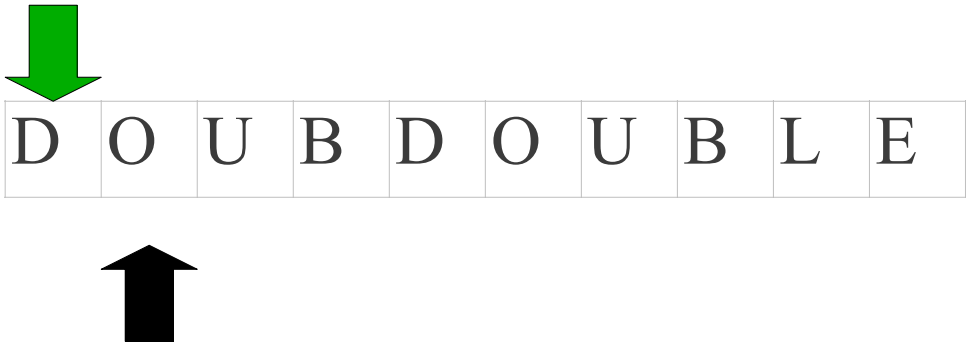




Implementing Maximal Munch

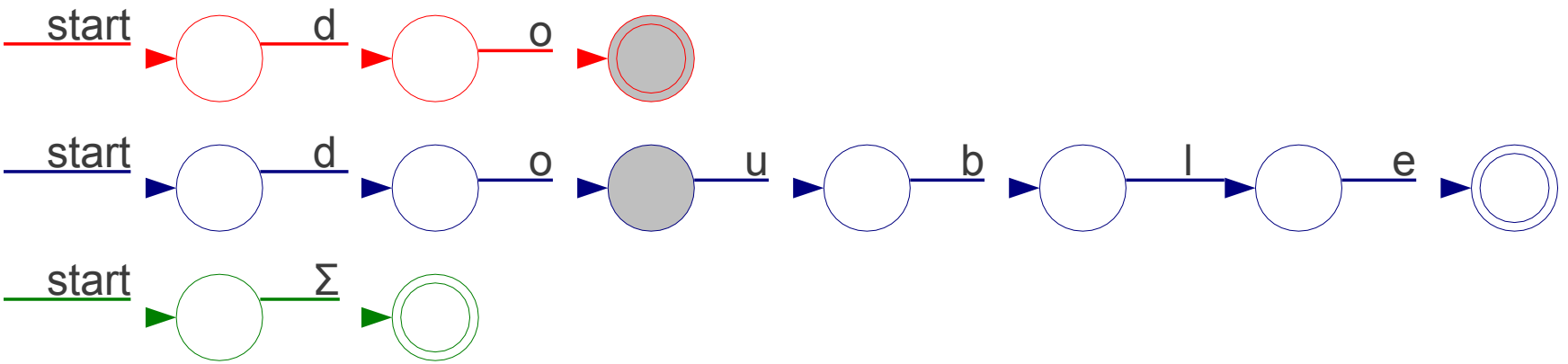
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

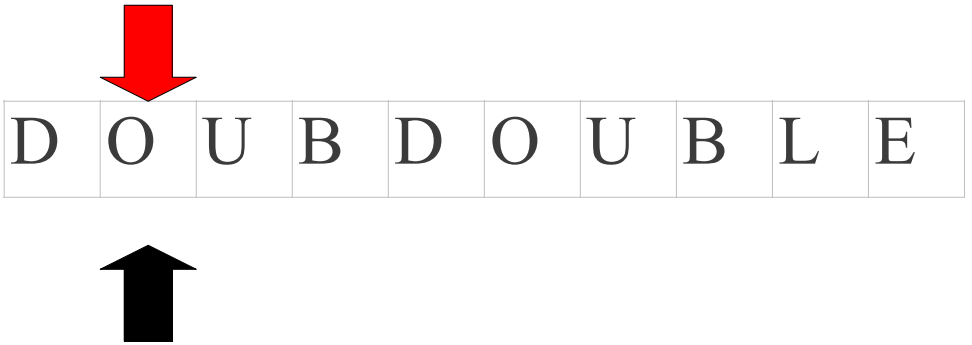




Implementing Maximal Munch

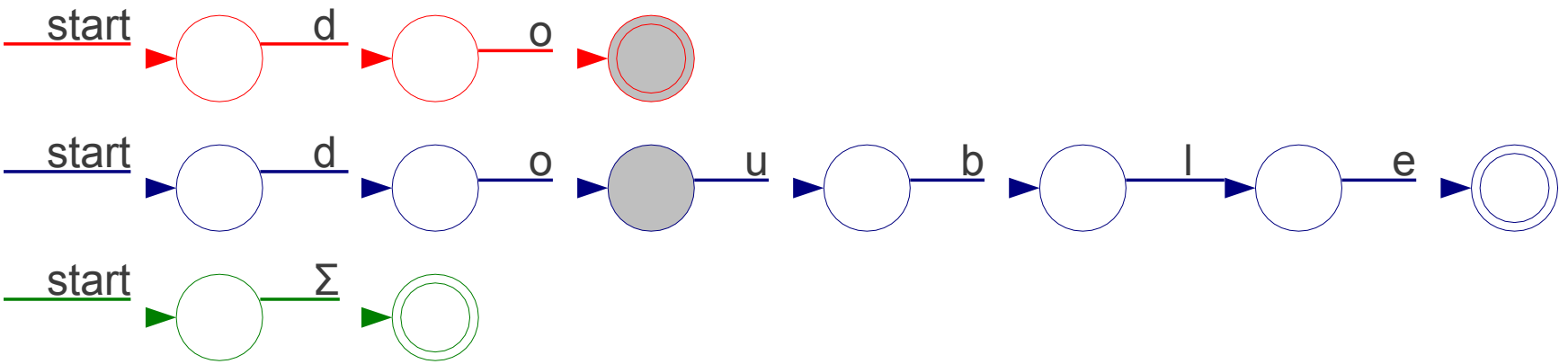
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

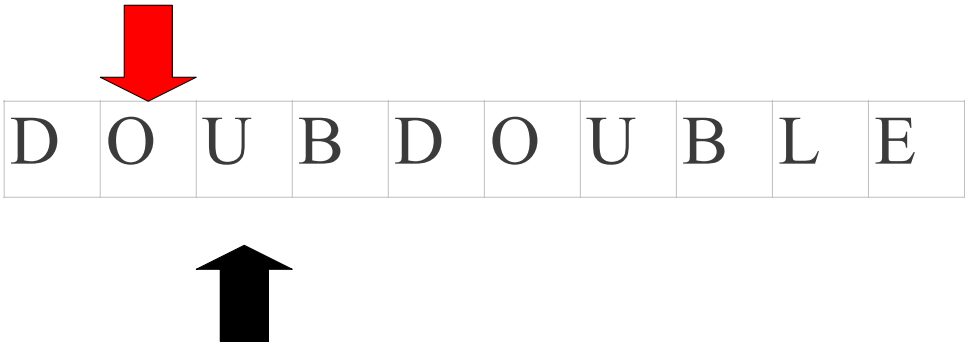




Implementing Maximal Munch

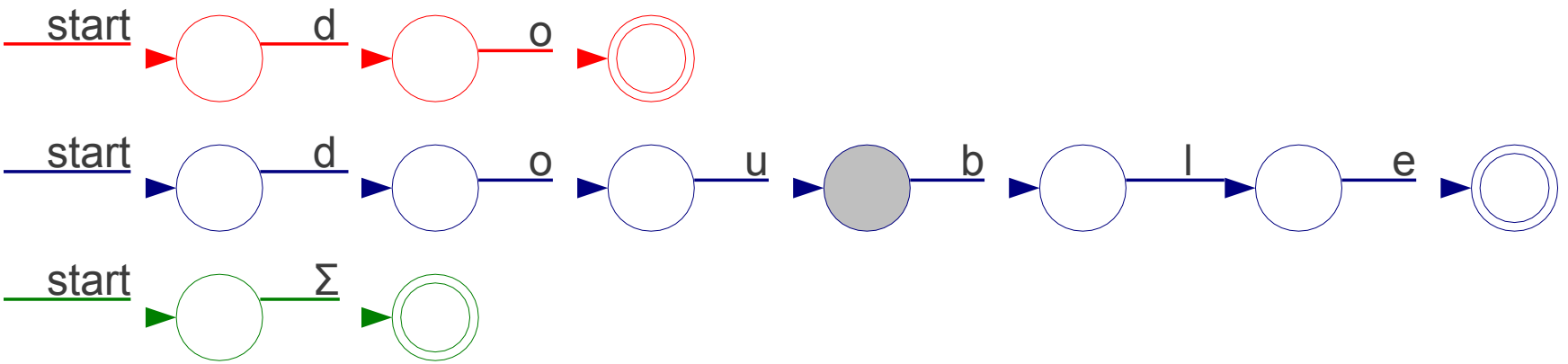
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

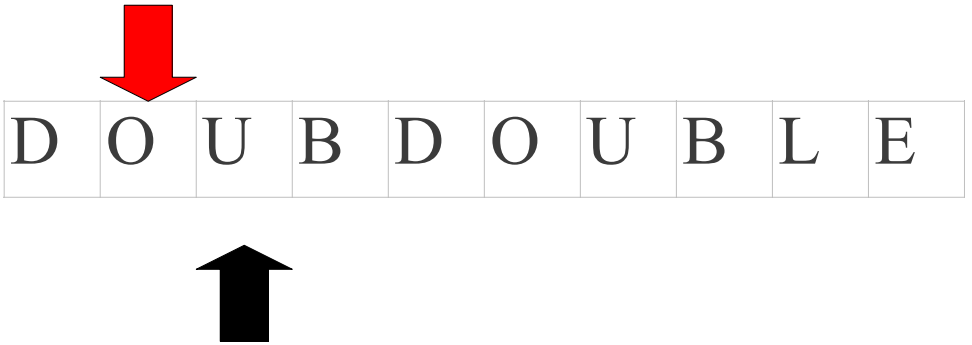




Implementing Maximal Munch

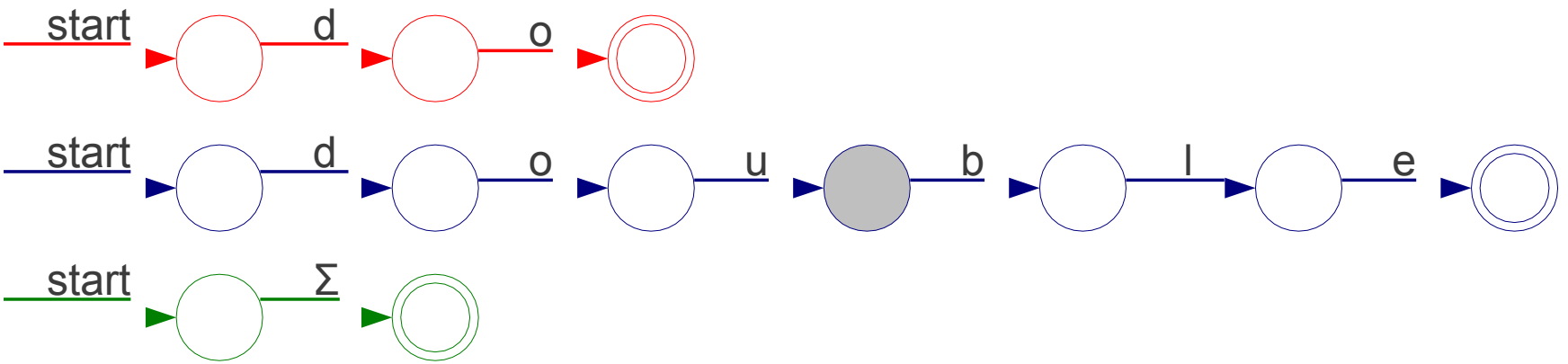
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

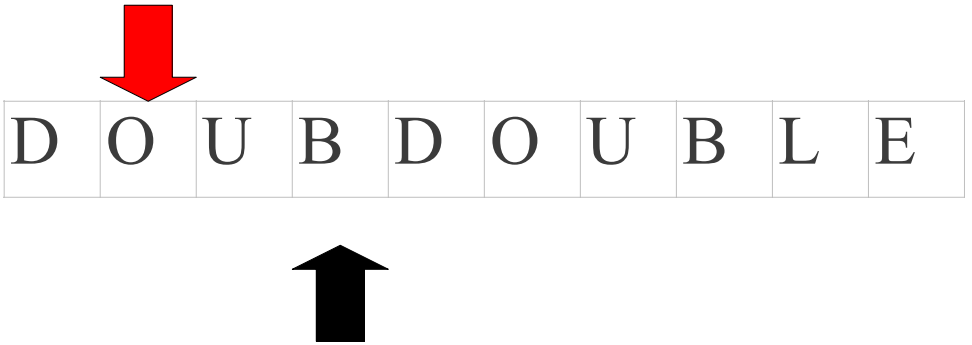




Implementing Maximal Munch

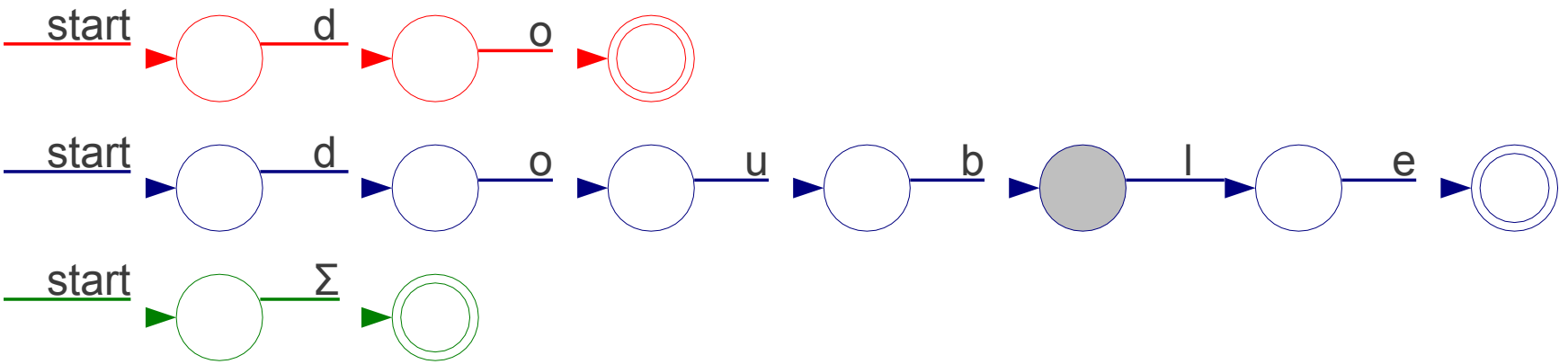
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

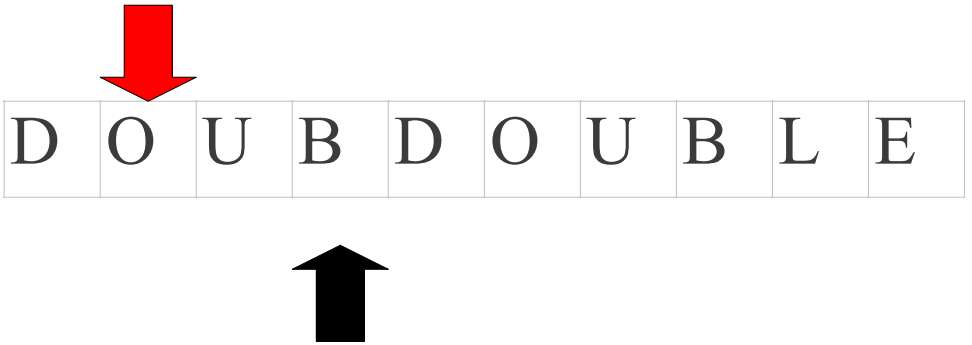




Implementing Maximal Munch

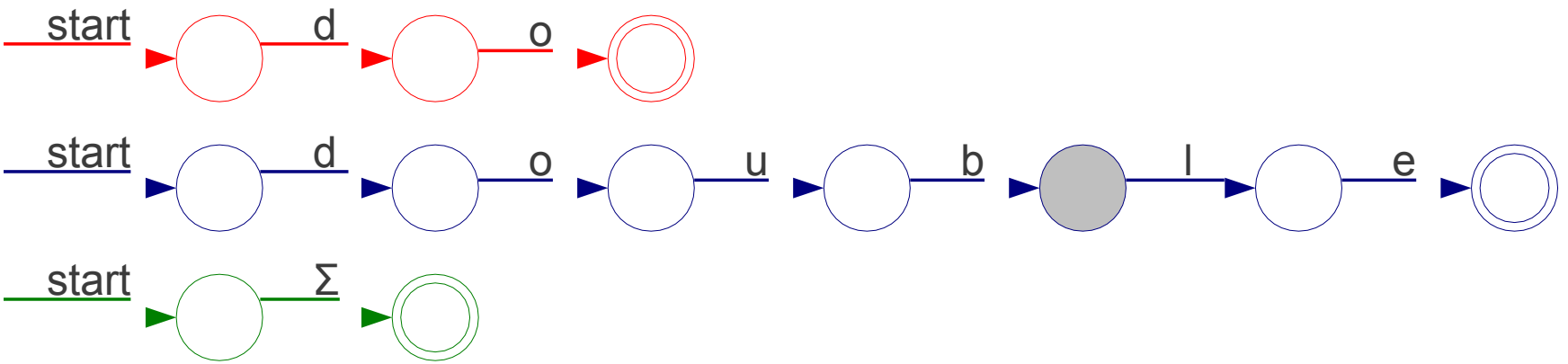
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

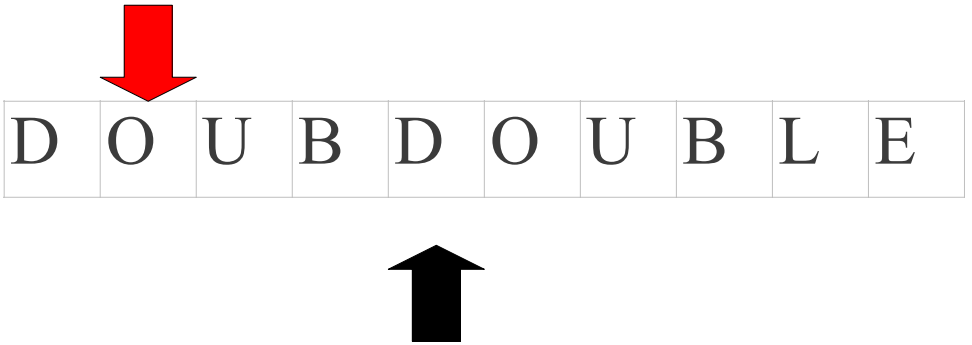




Implementing Maximal Munch

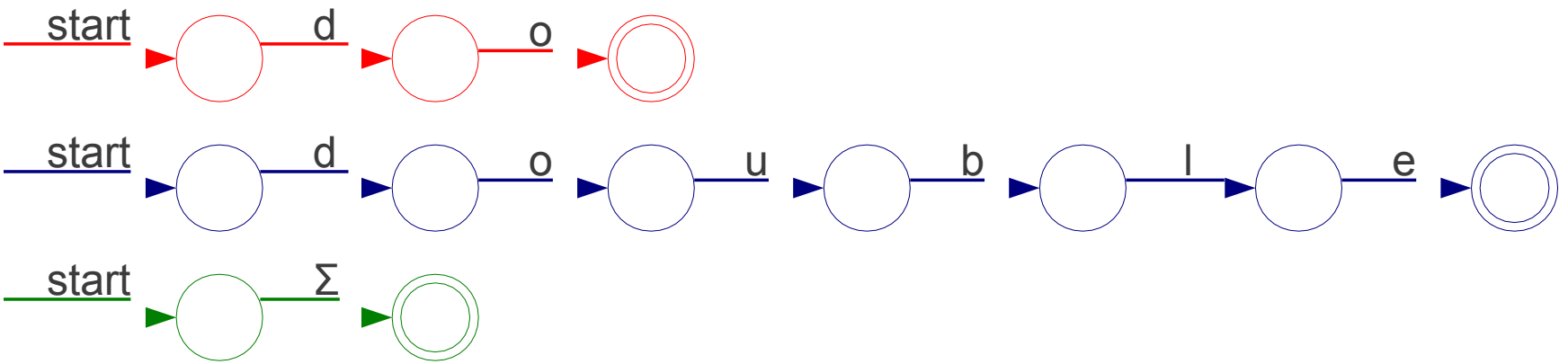
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

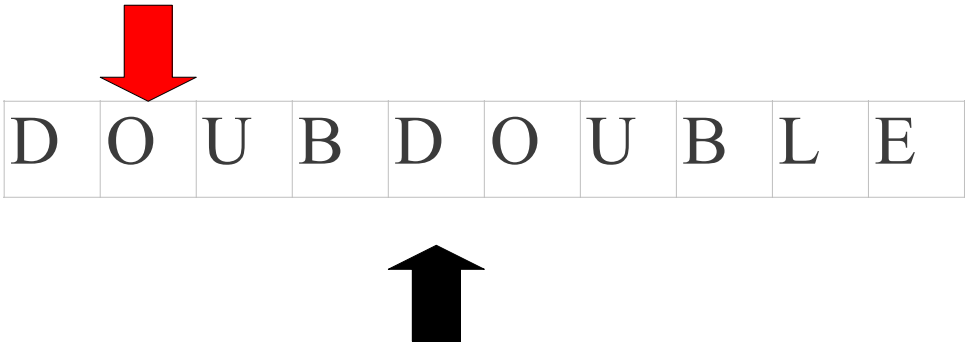




Implementing Maximal Munch

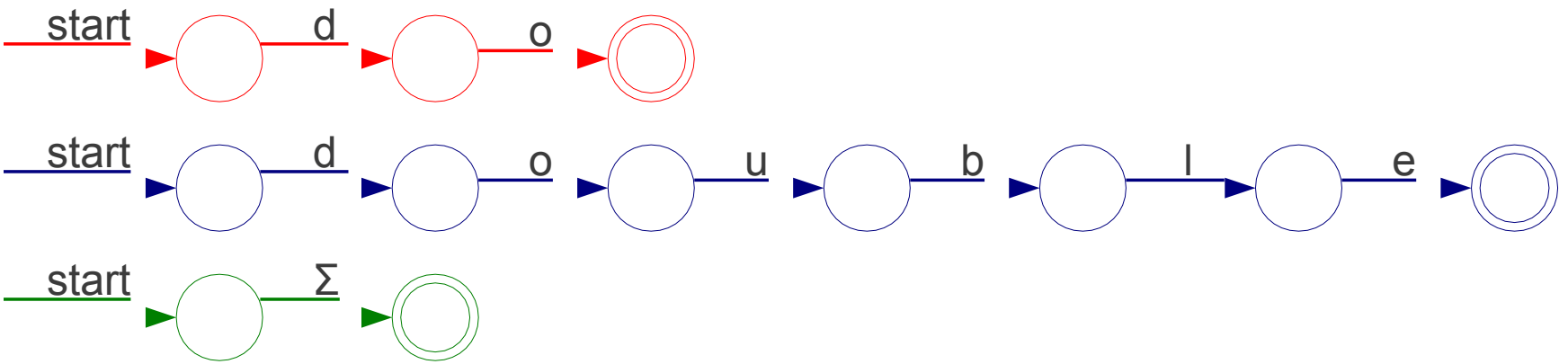
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```





Implementing Maximal Munch

```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

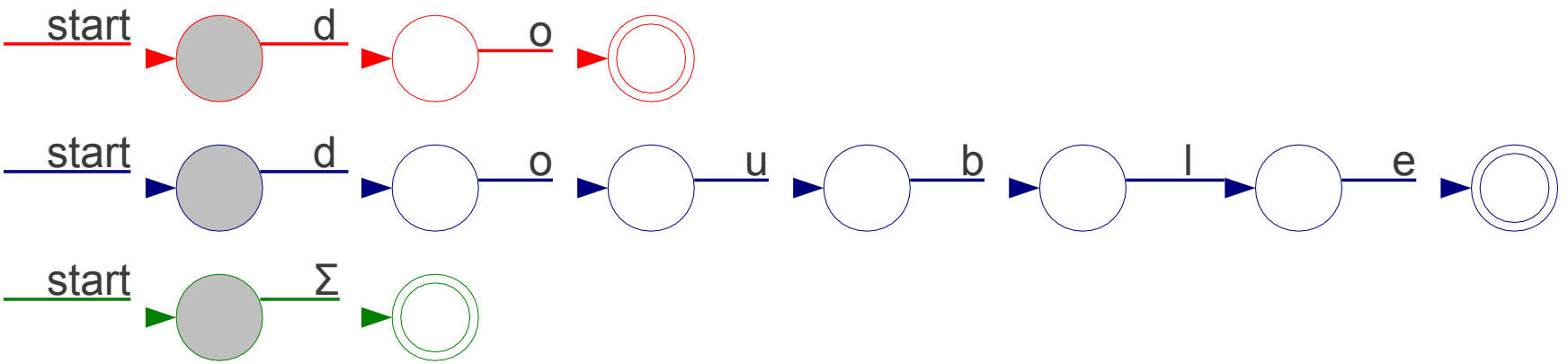


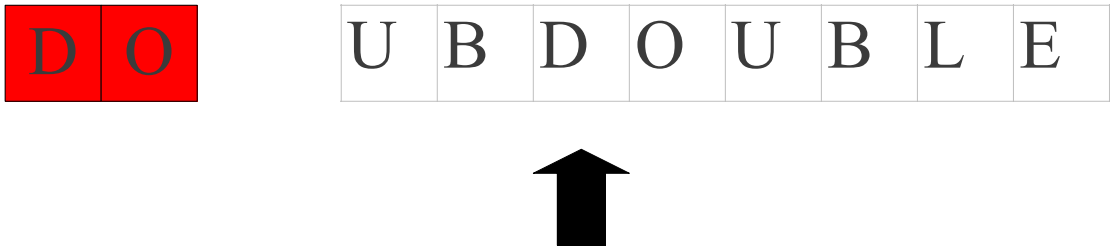
D	O

U	B	D	O	U	B	L	E

Implementing Maximal Munch

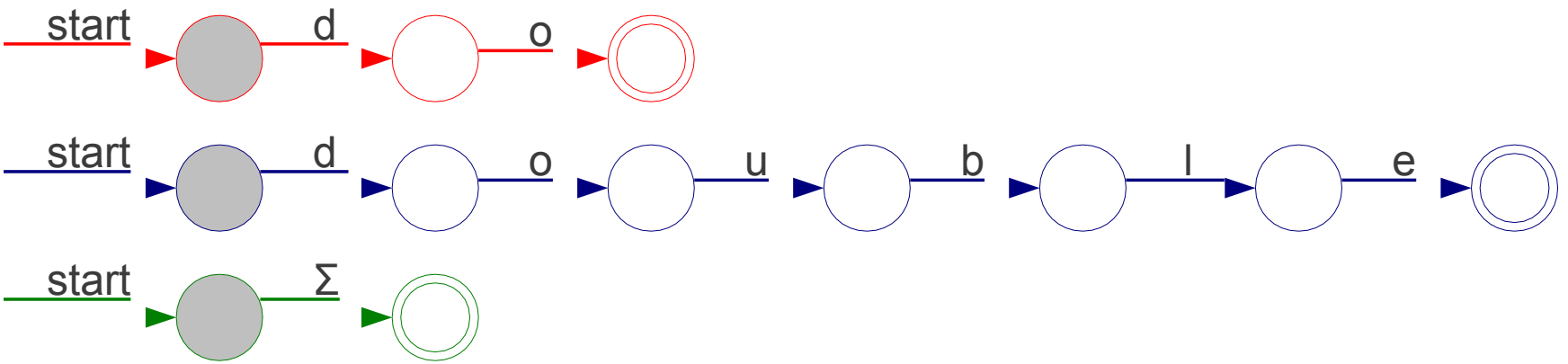
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

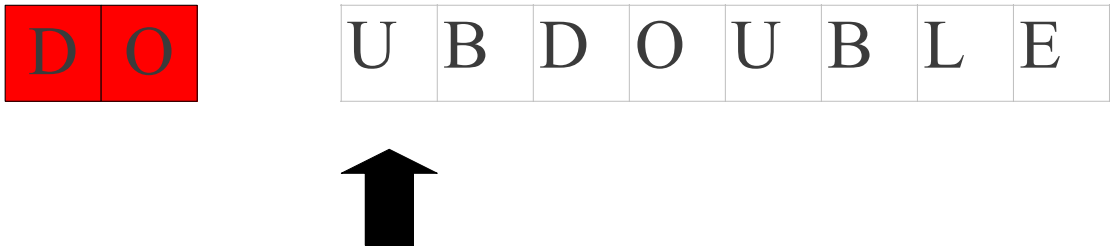




Implementing Maximal Munch

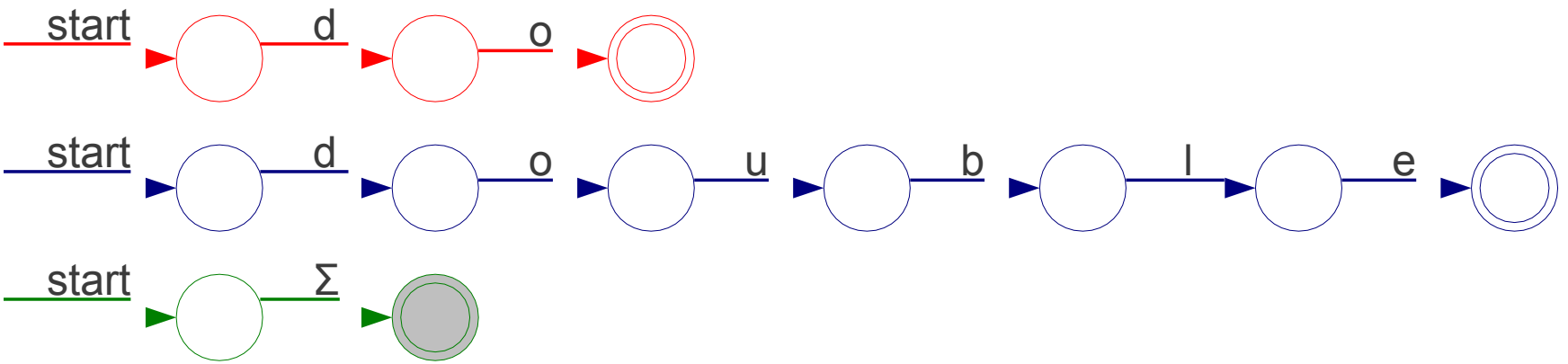
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

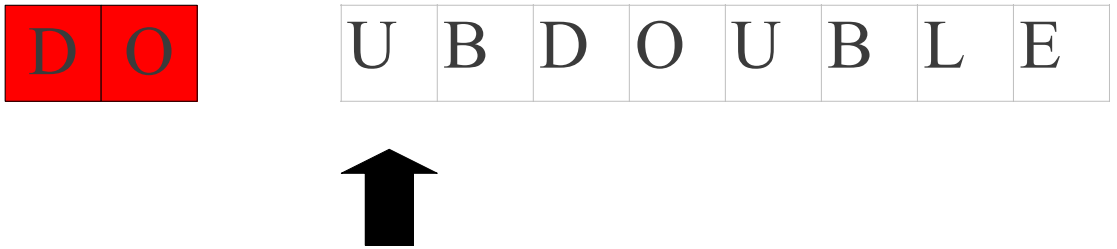




Implementing Maximal Munch

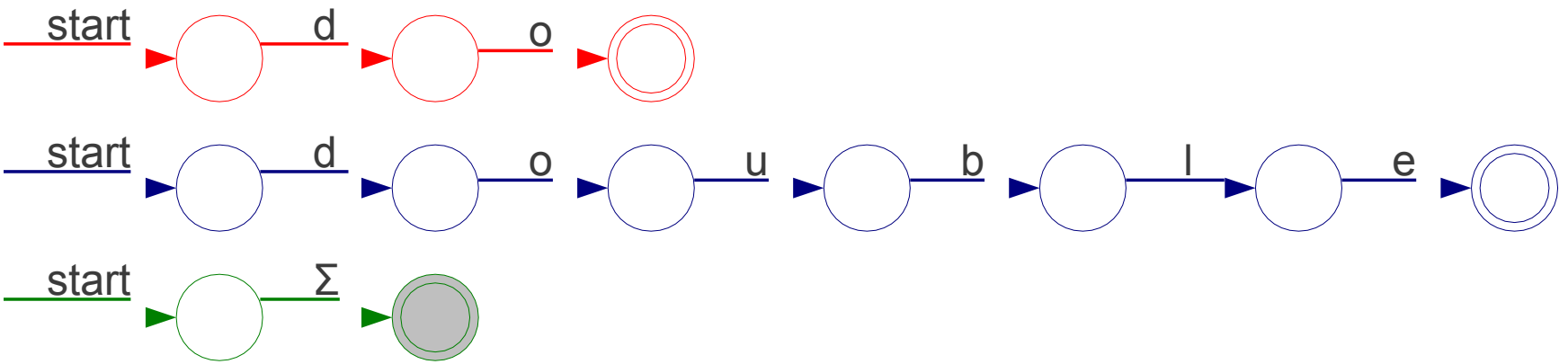
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

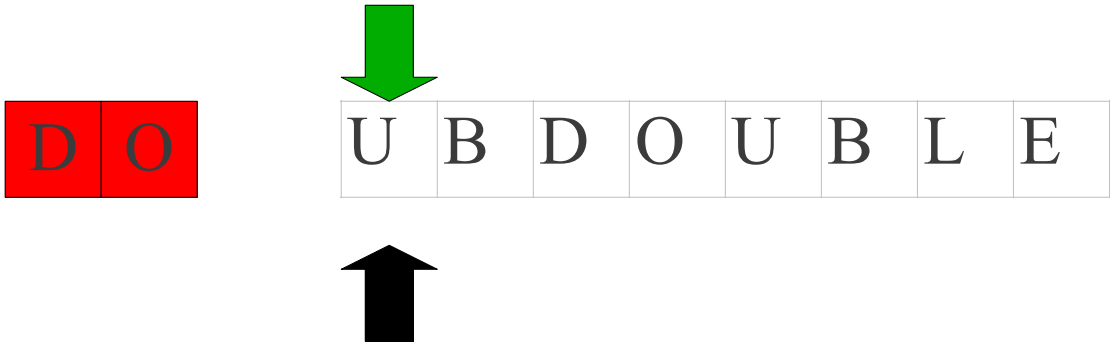




Implementing Maximal Munch

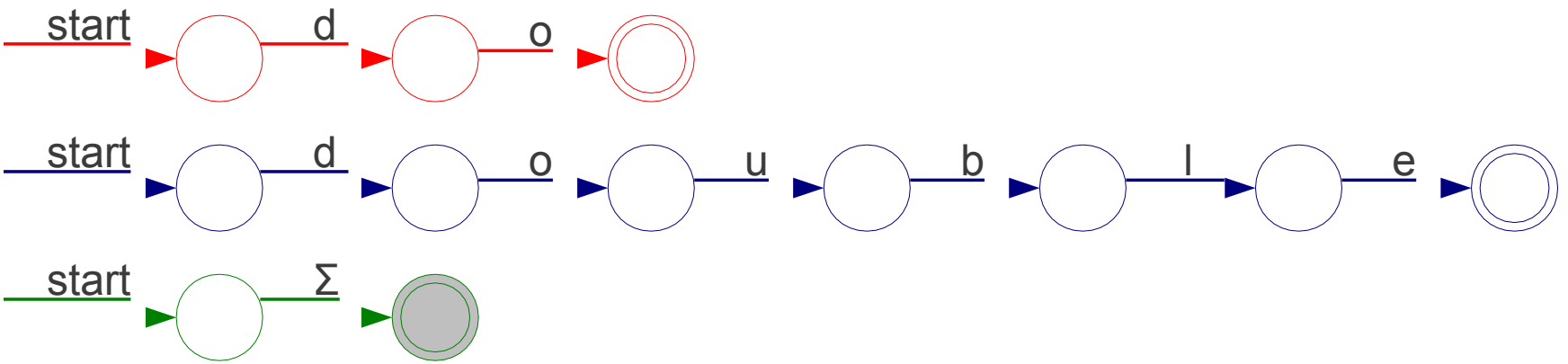
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

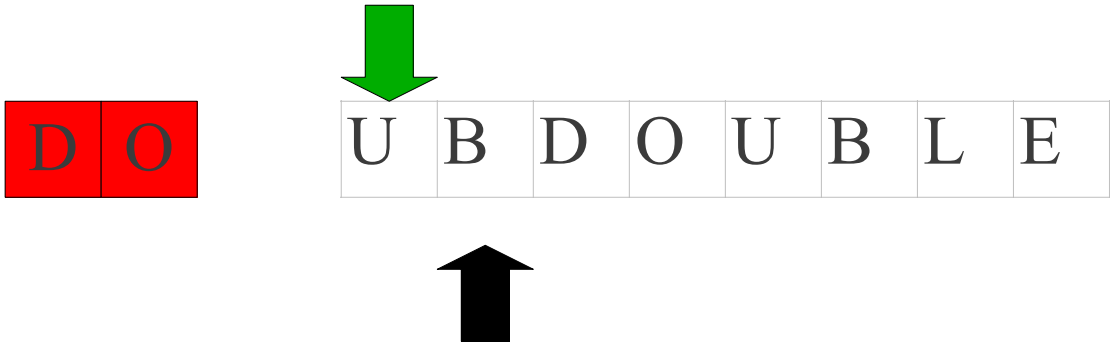




Implementing Maximal Munch

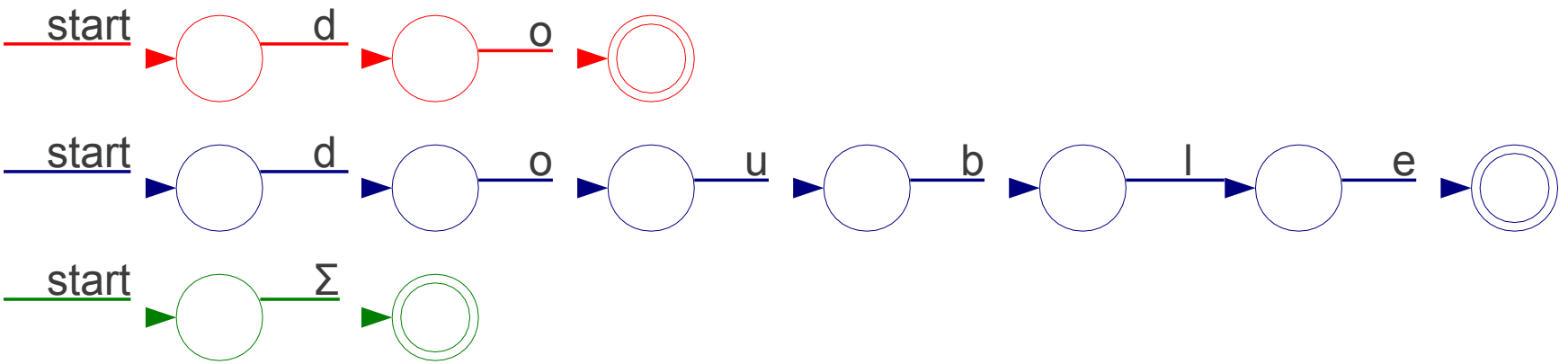
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

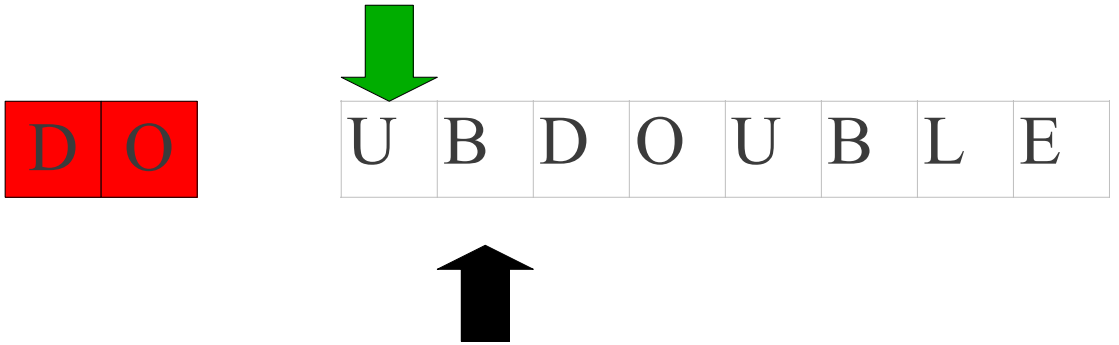




Implementing Maximal Munch

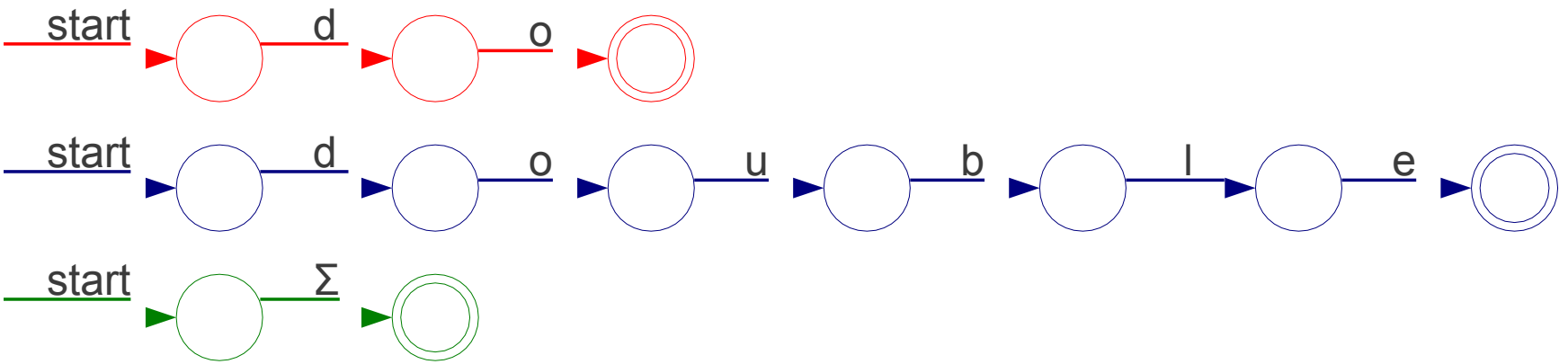
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

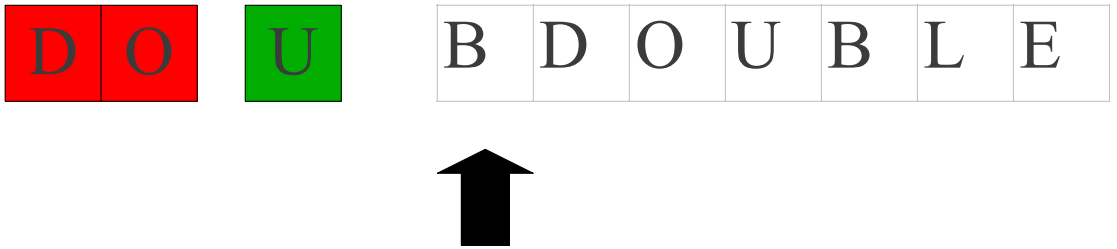




Implementing Maximal Munch

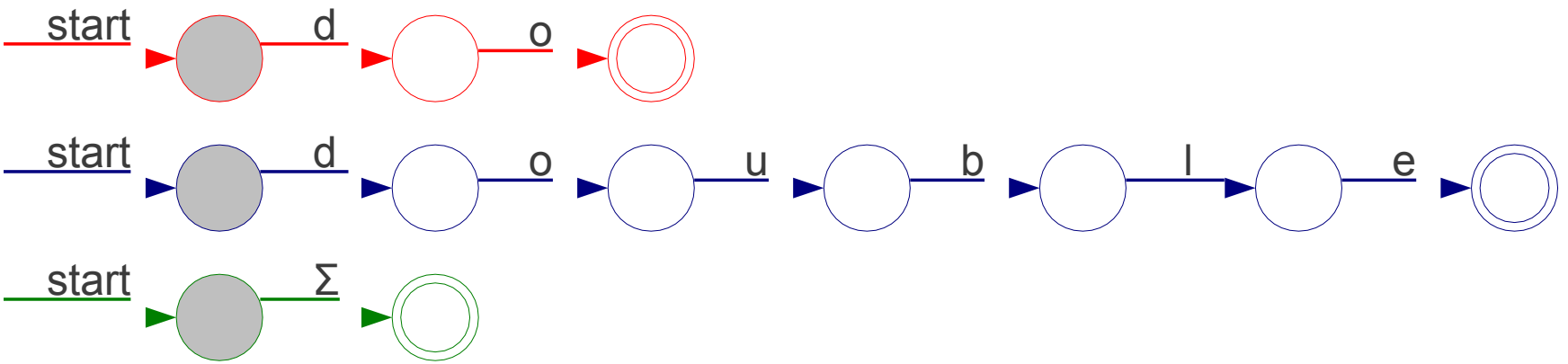
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

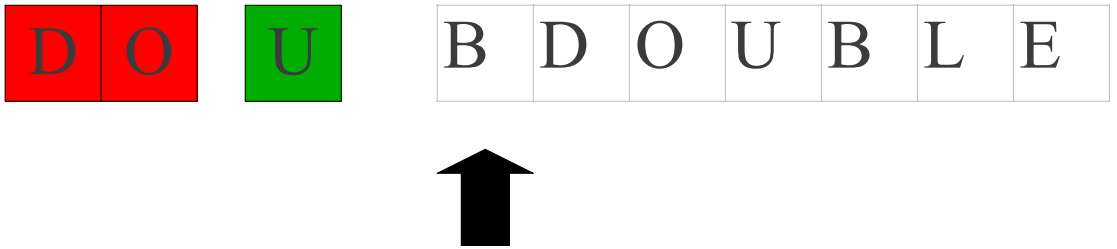




Implementing Maximal Munch

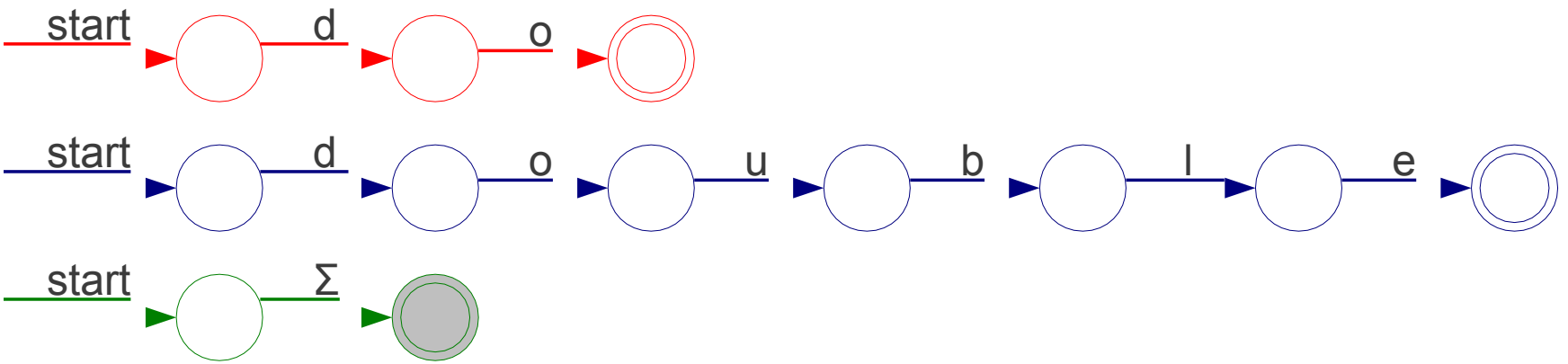
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

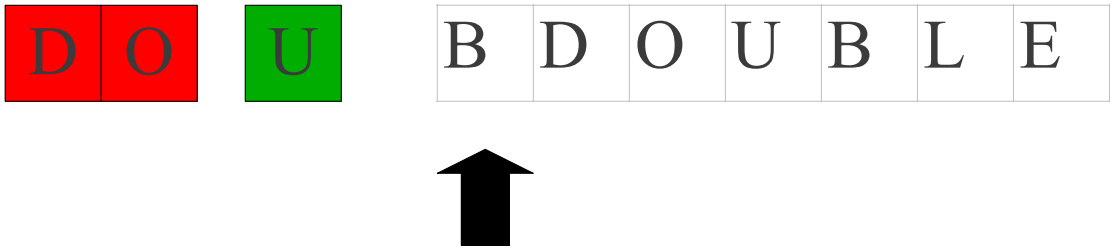




Implementing Maximal Munch

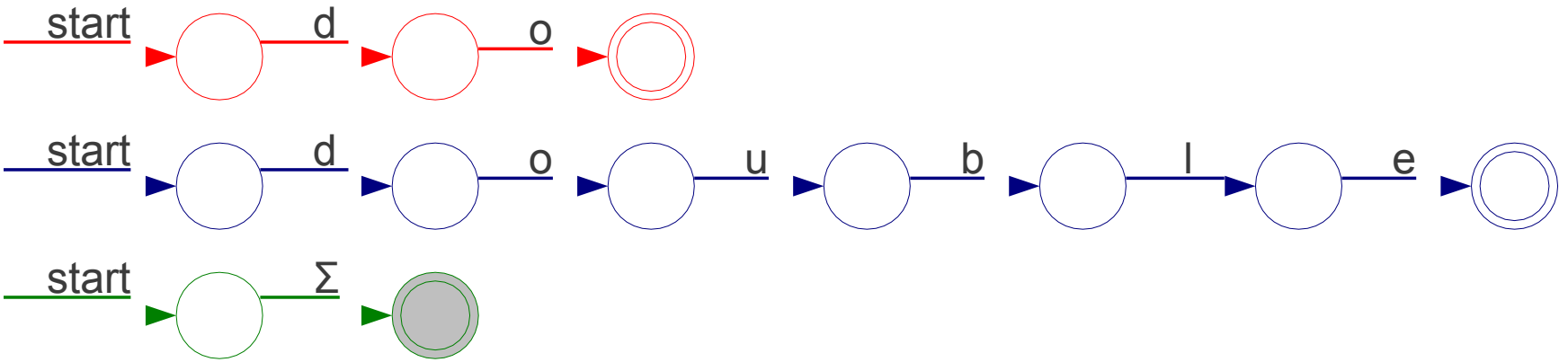
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

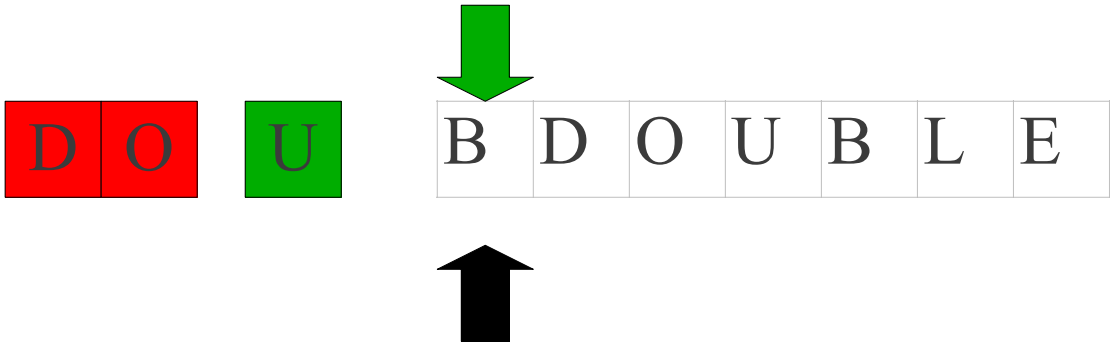




Implementing Maximal Munch

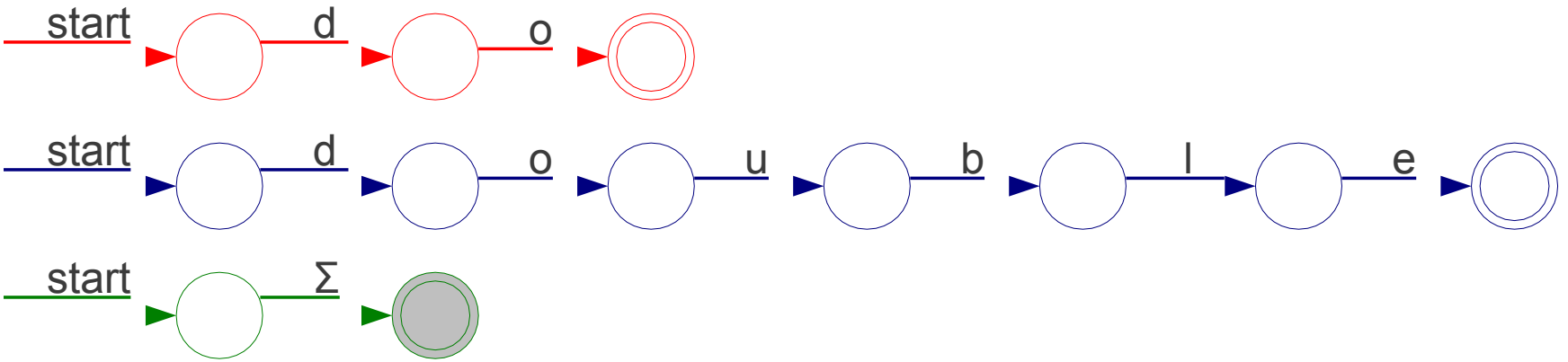
```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

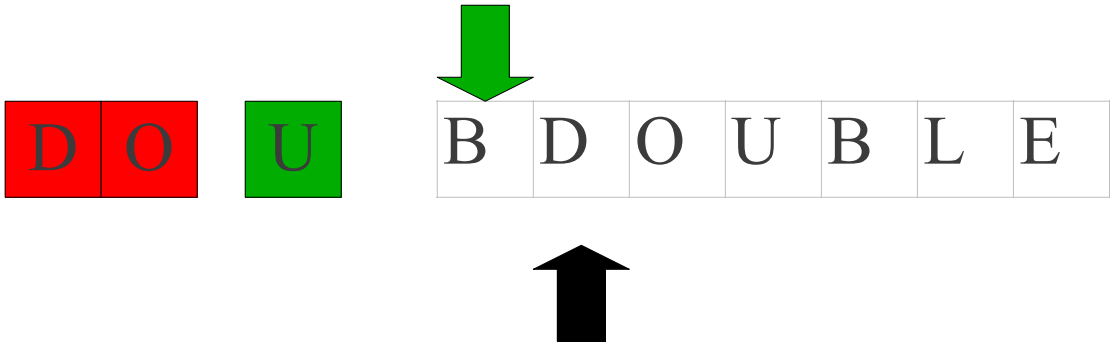




Implementing Maximal Munch

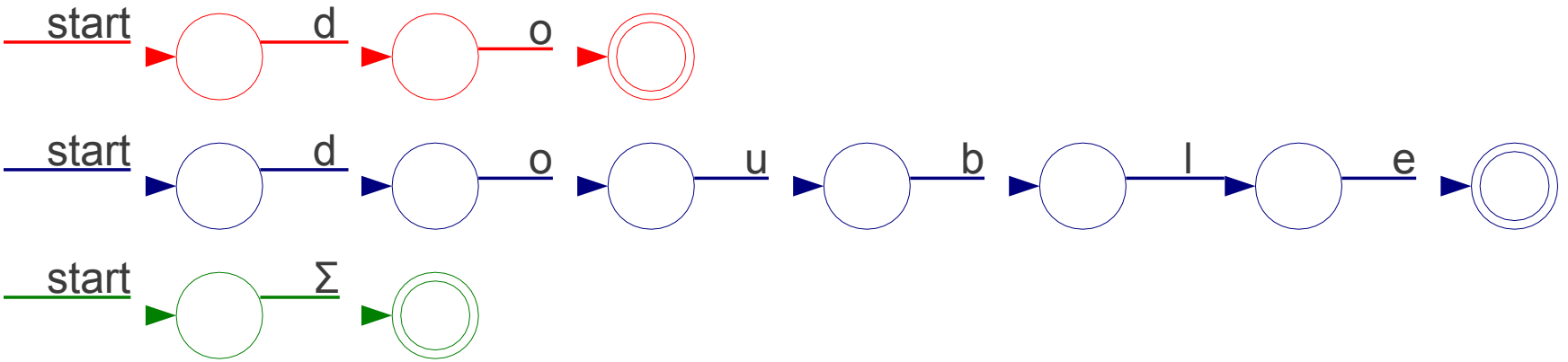
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

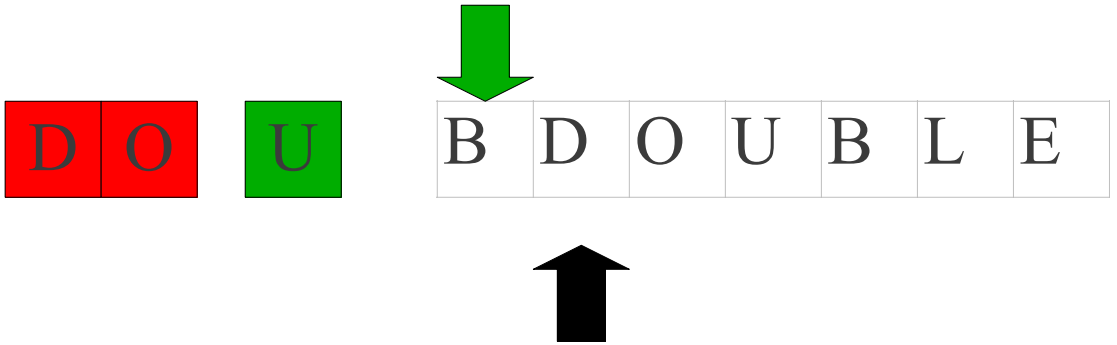




Implementing Maximal Munch

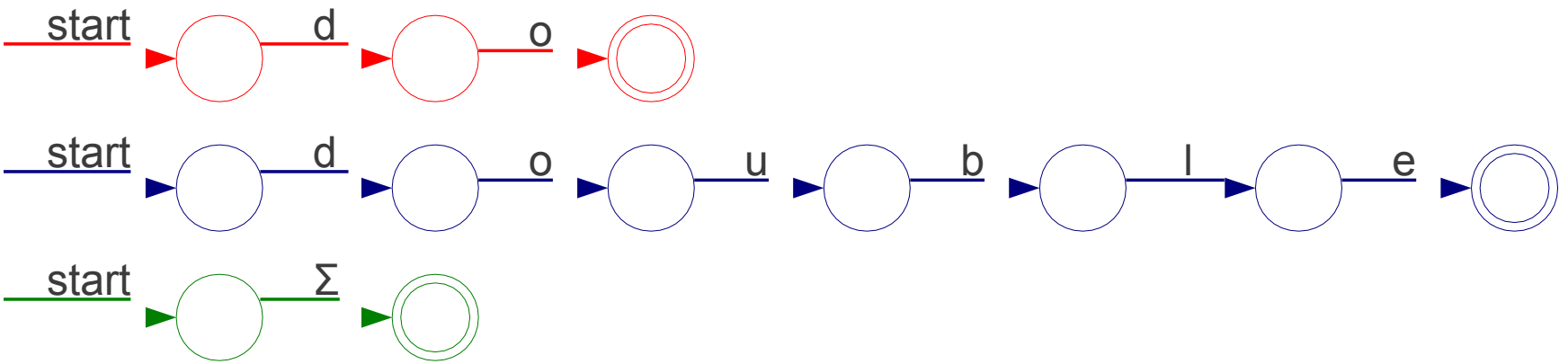
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

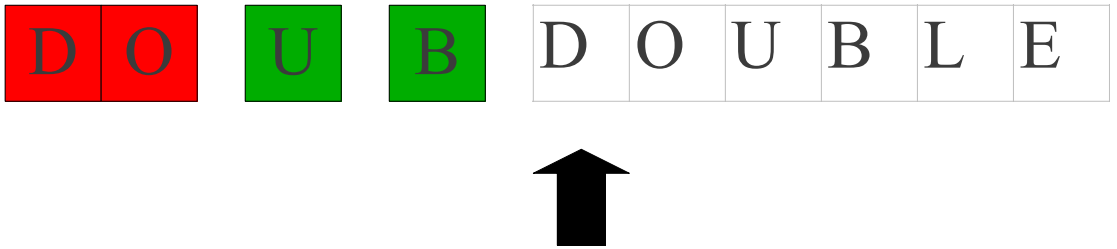




Implementing Maximal Munch

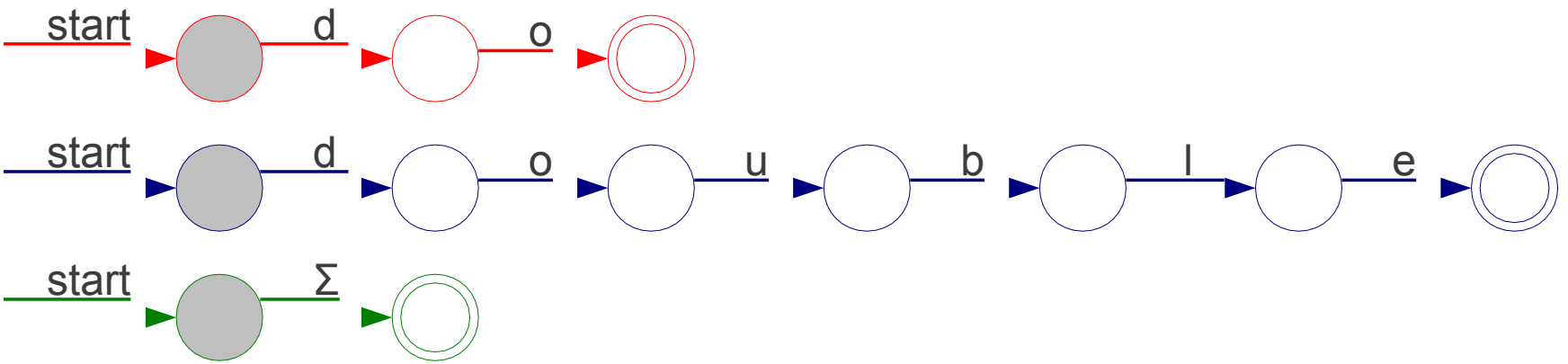
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

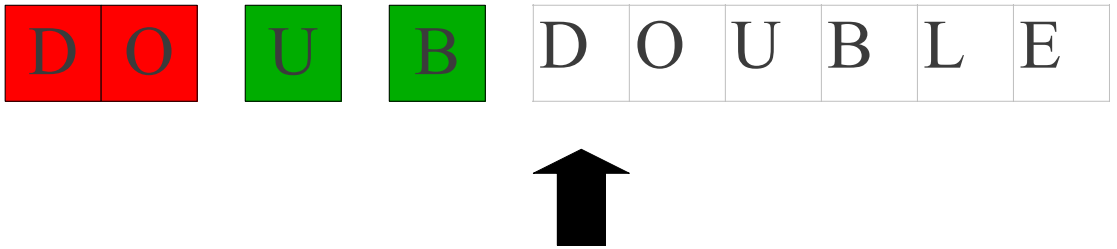




Implementing Maximal Munch

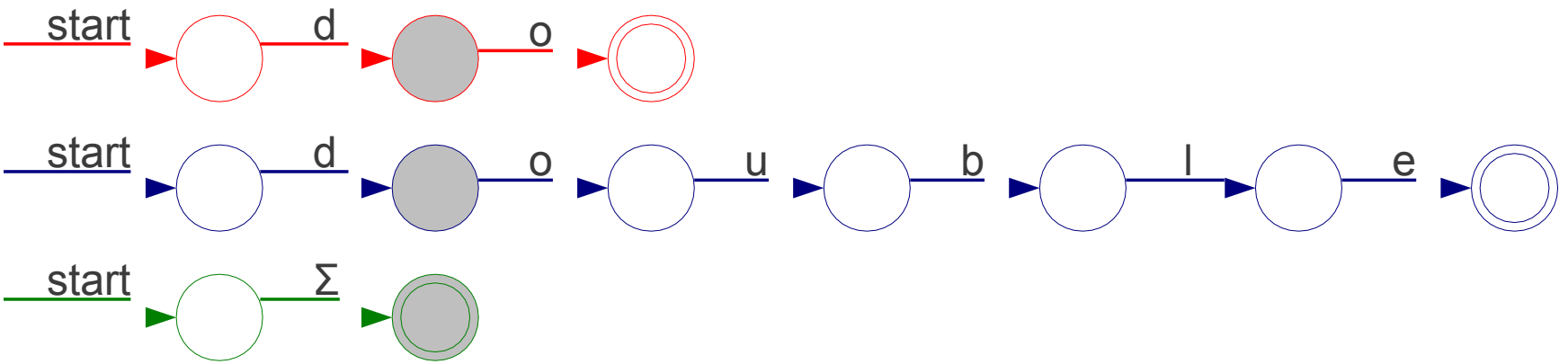
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

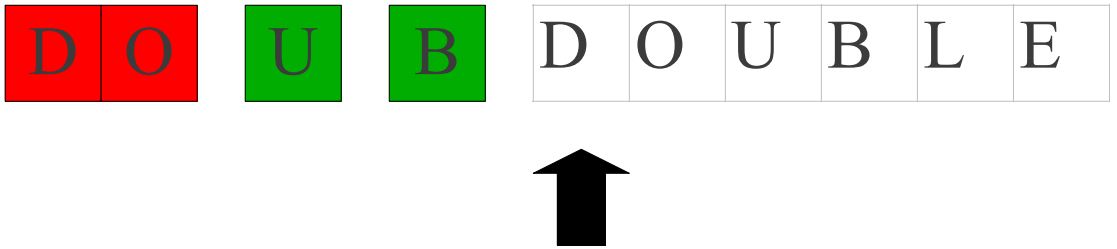




Implementing Maximal Munch

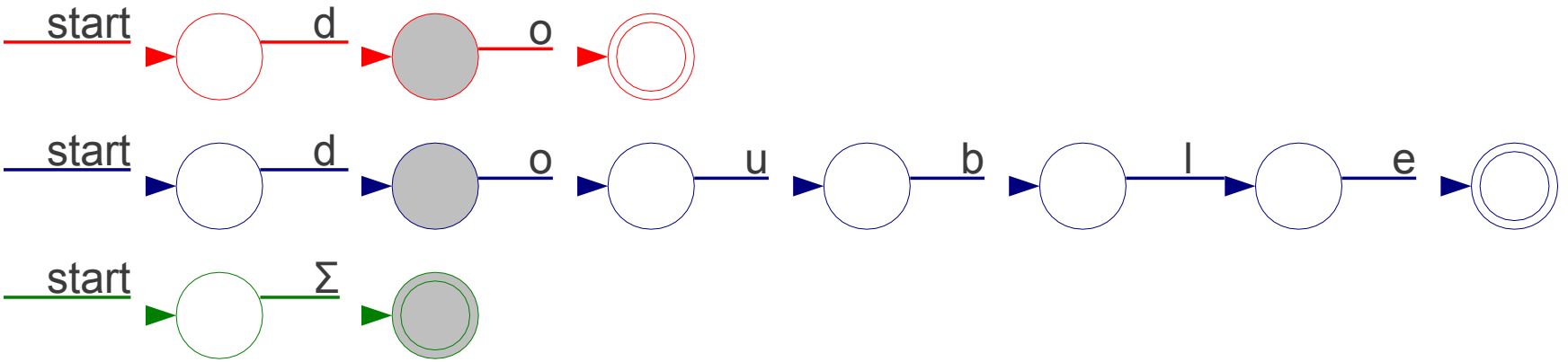
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

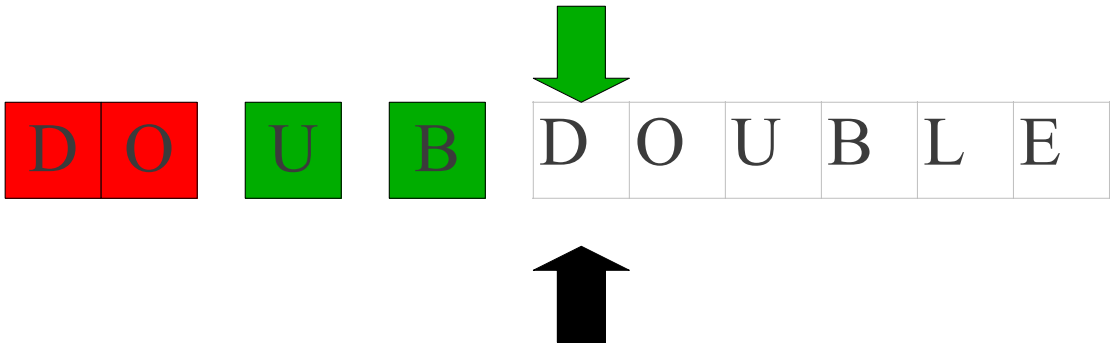




Implementing Maximal Munch

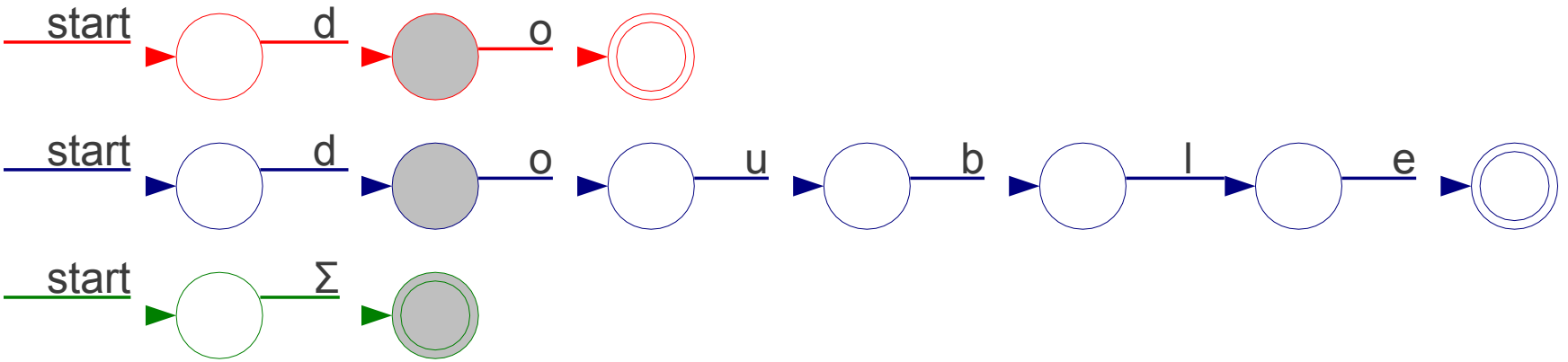
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

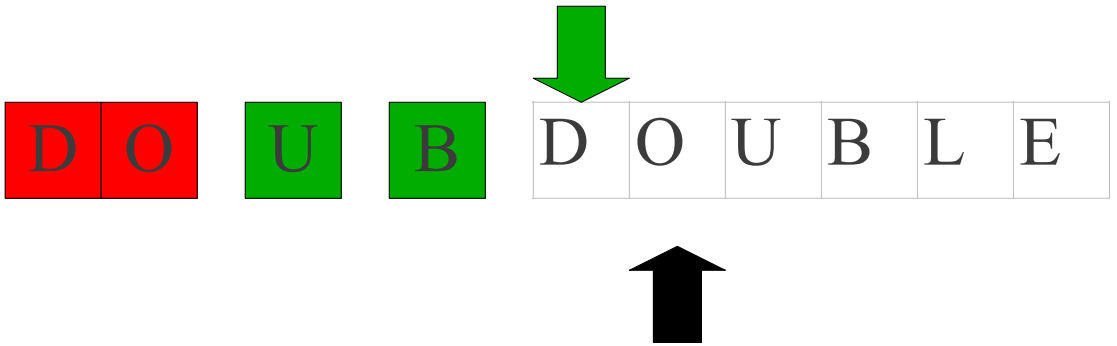




Implementing Maximal Munch

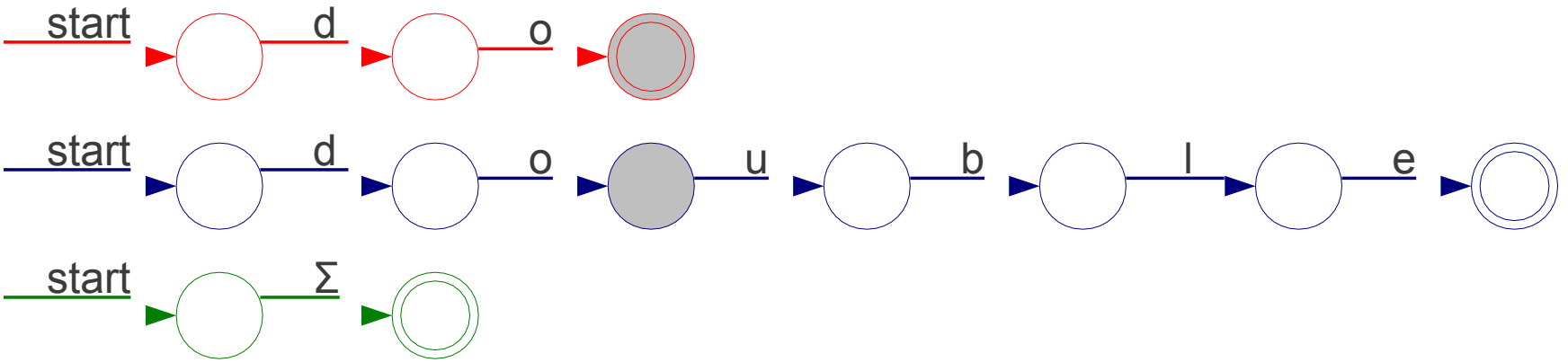
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

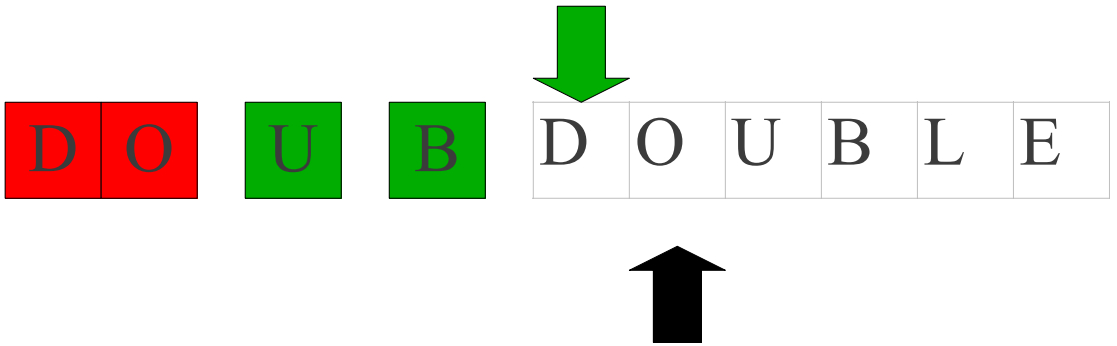




Implementing Maximal Munch

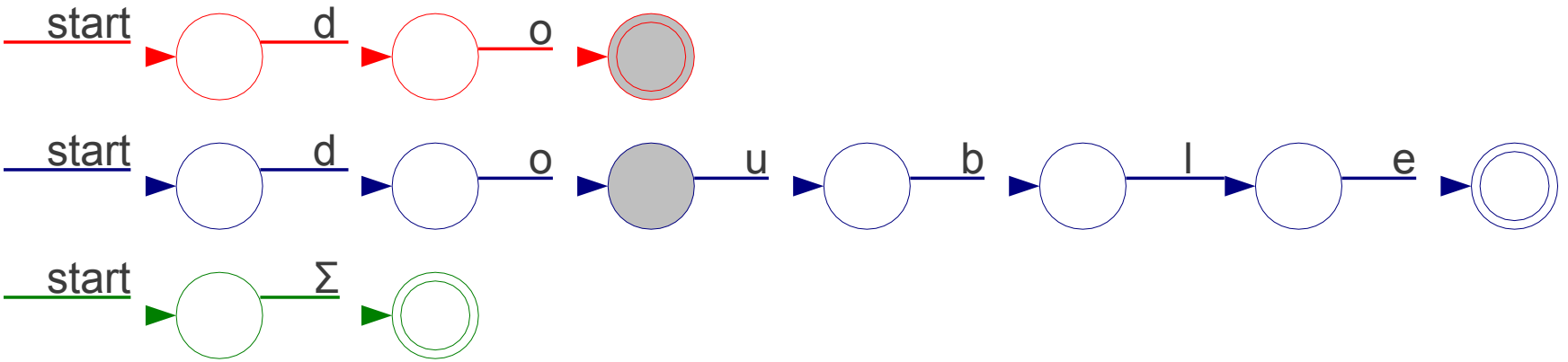
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

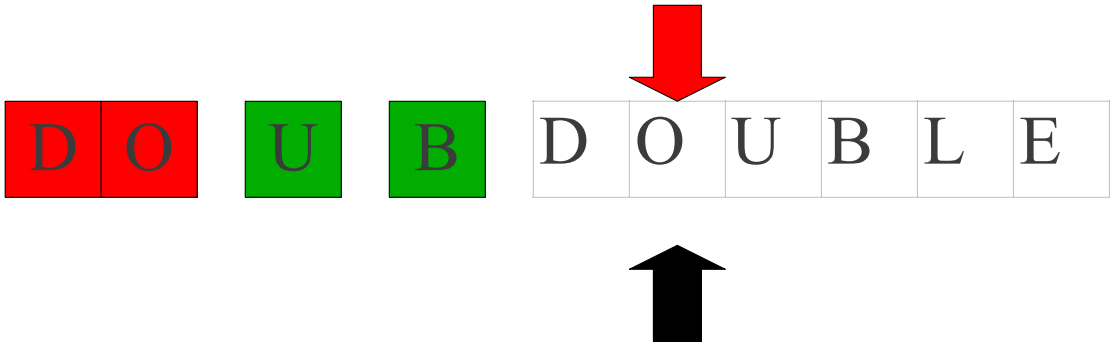




Implementing Maximal Munch

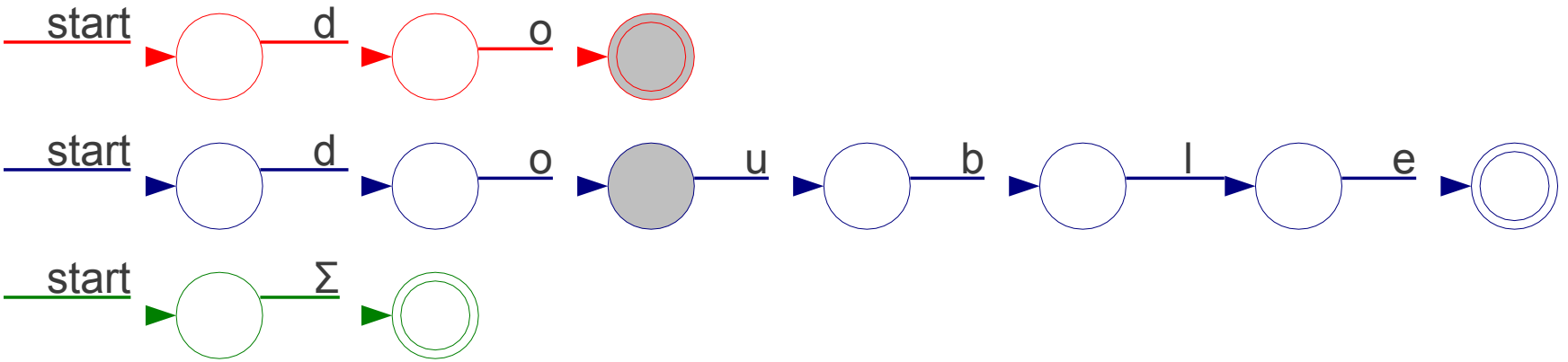
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

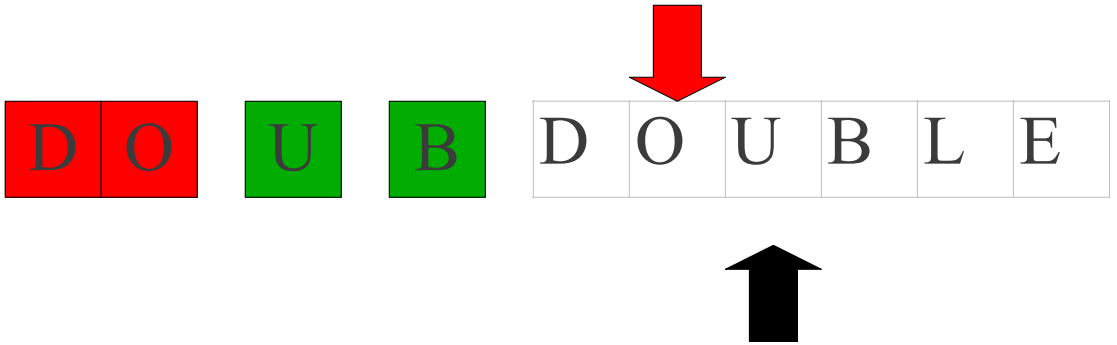




Implementing Maximal Munch

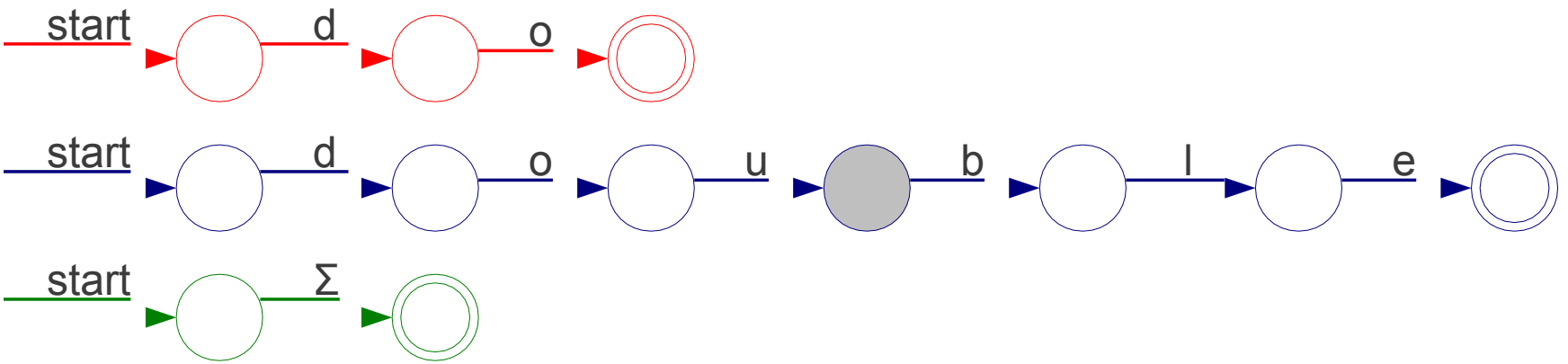
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

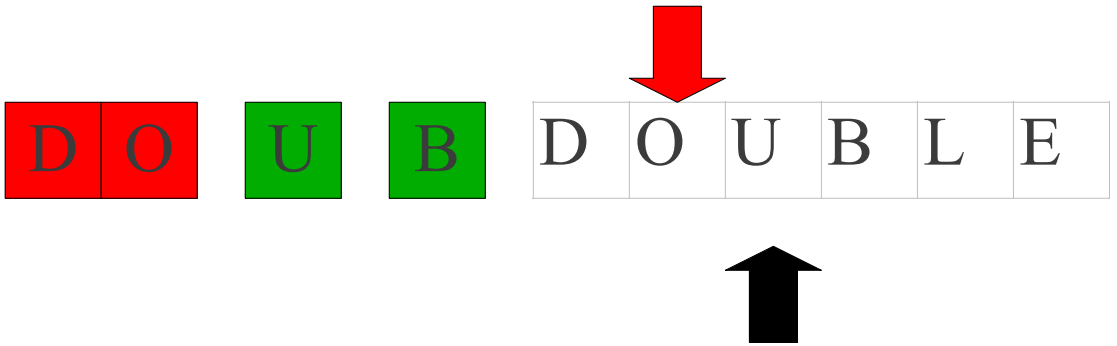




Implementing Maximal Munch

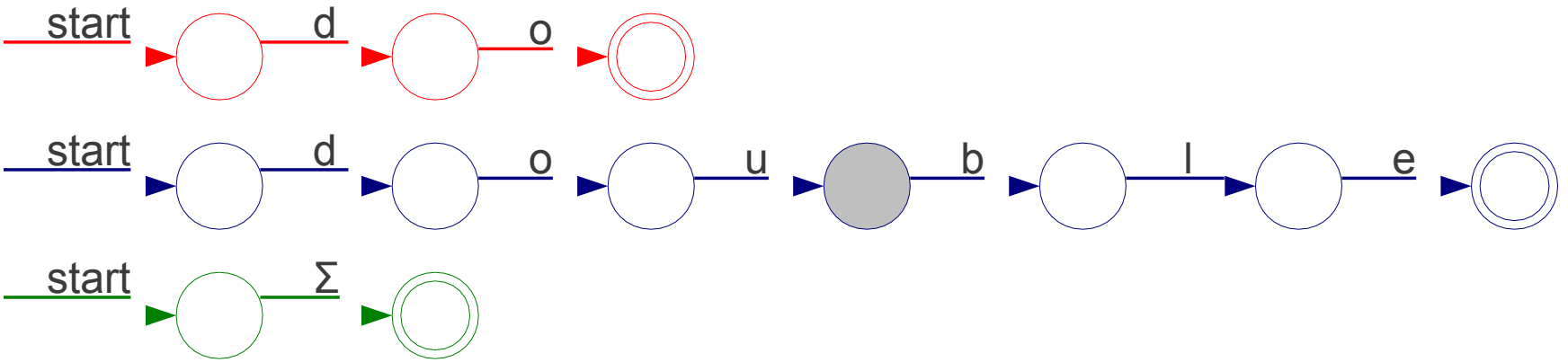
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

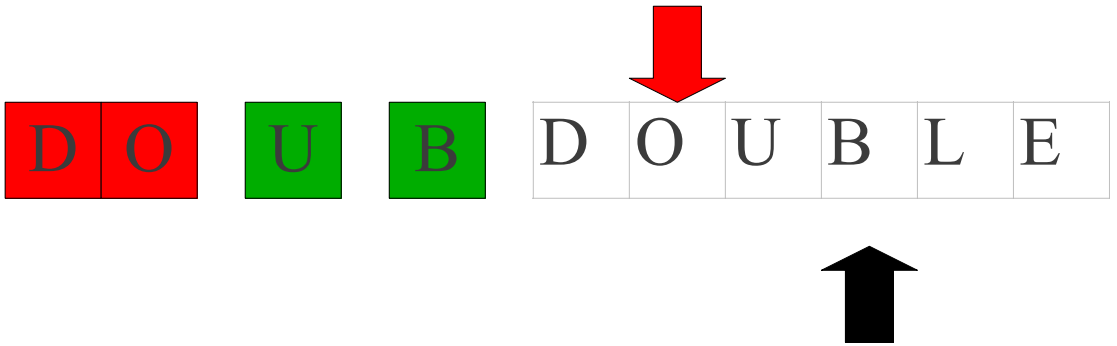




Implementing Maximal Munch

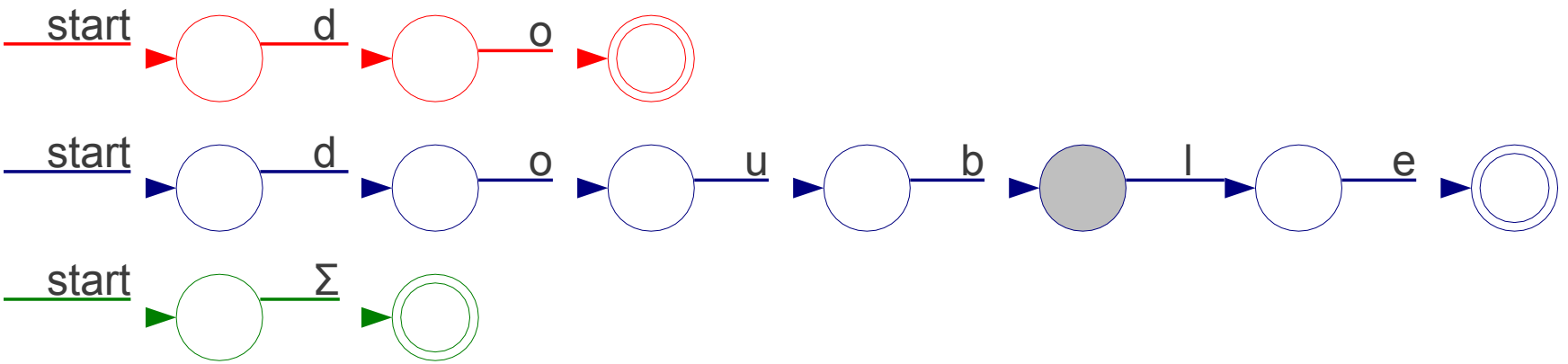
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

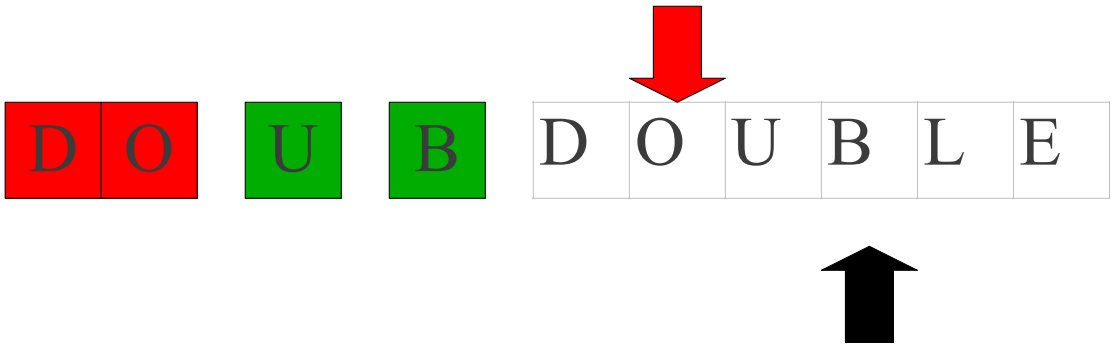




Implementing Maximal Munch

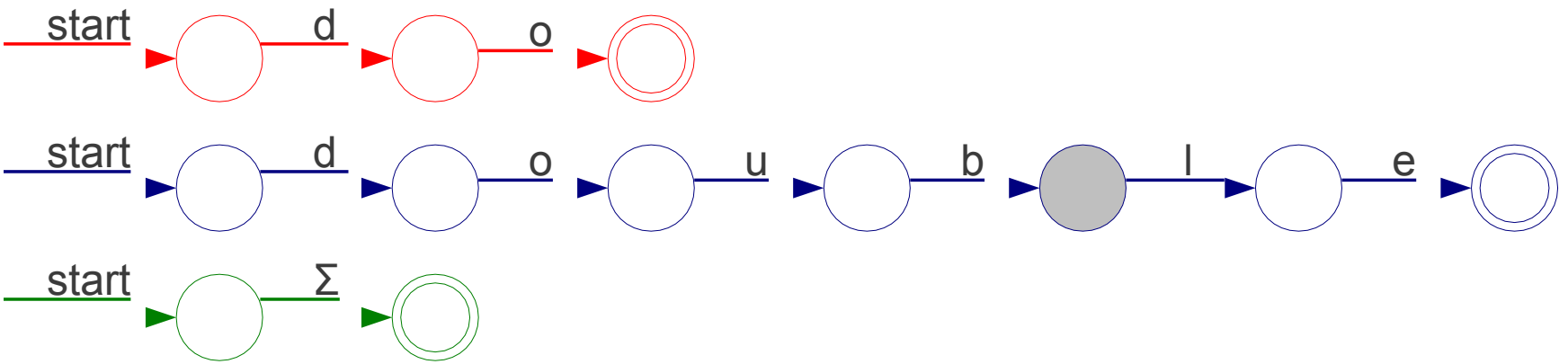
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

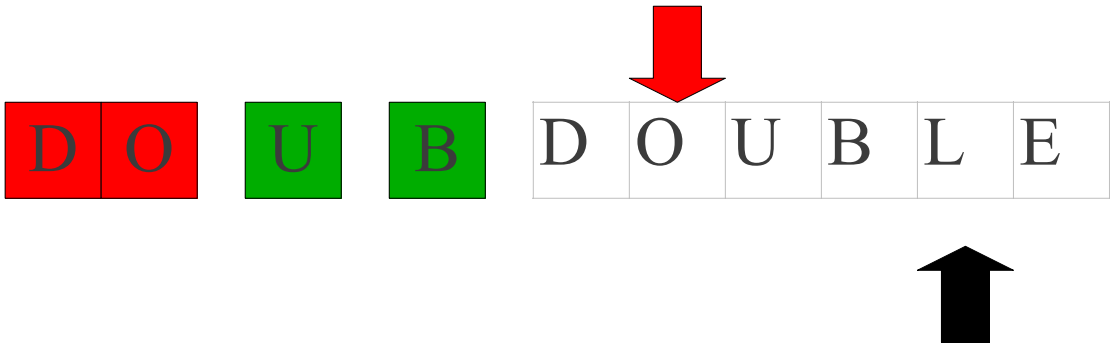




Implementing Maximal Munch

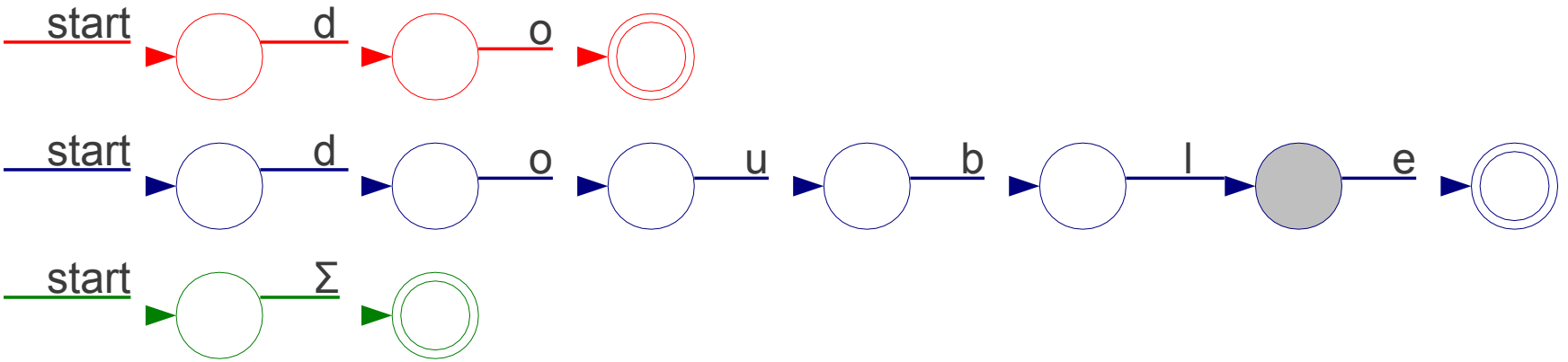
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

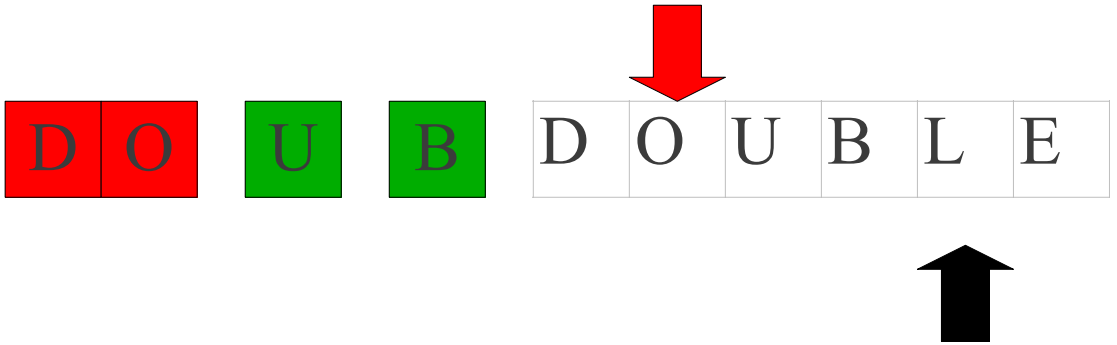




Implementing Maximal Munch

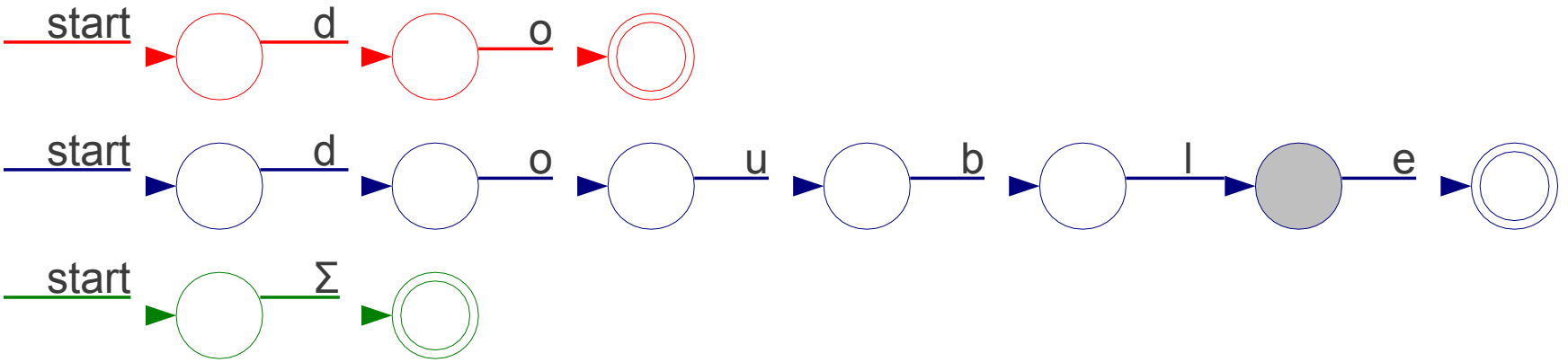
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

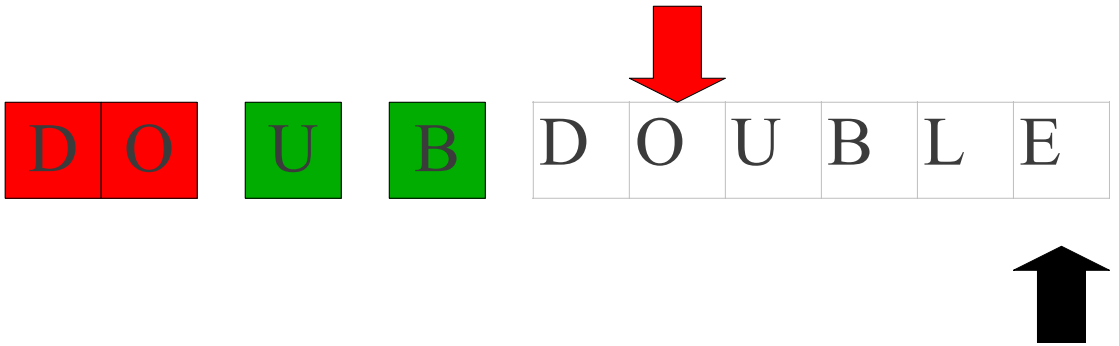




Implementing Maximal Munch

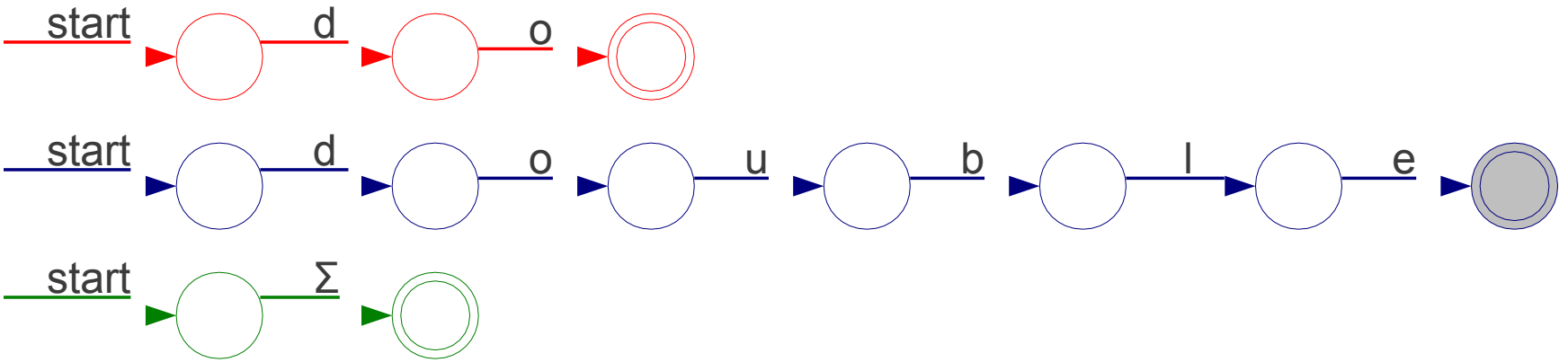
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

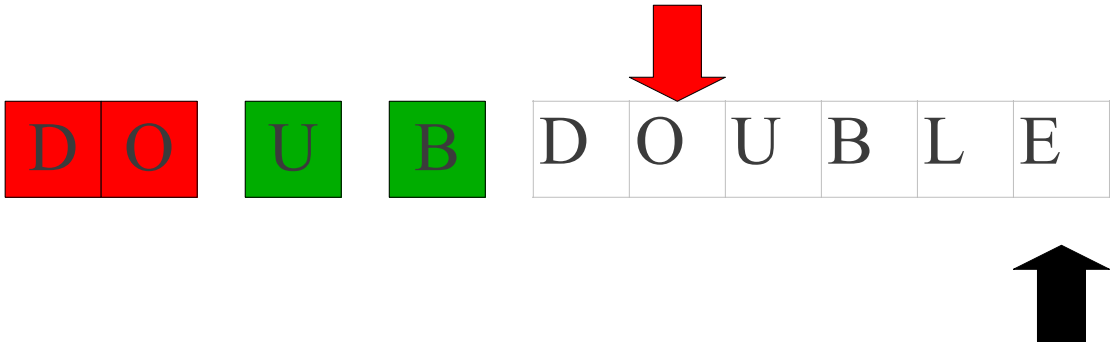




Implementing Maximal Munch

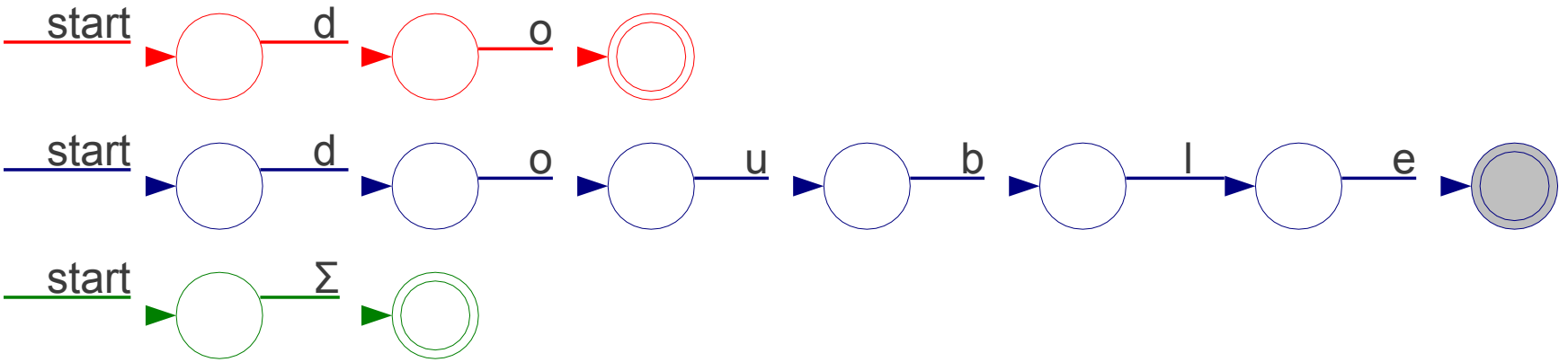
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

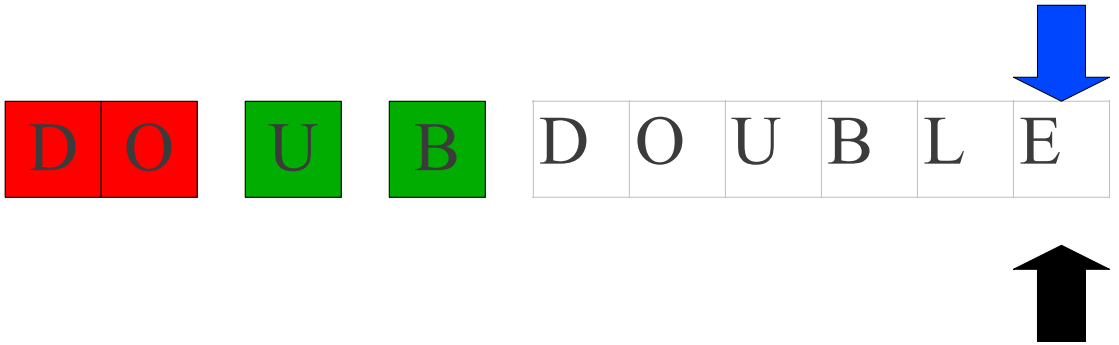




Implementing Maximal Munch

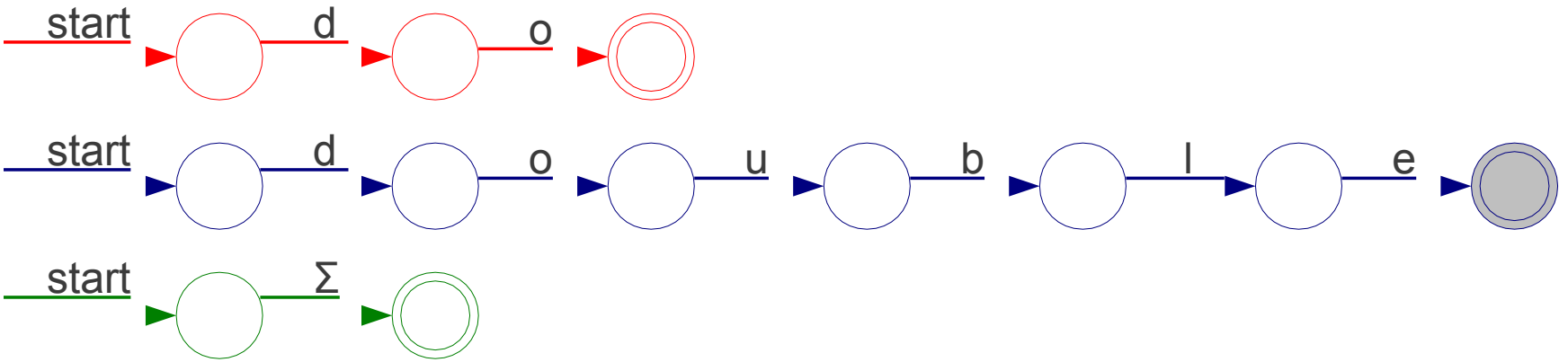
```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```





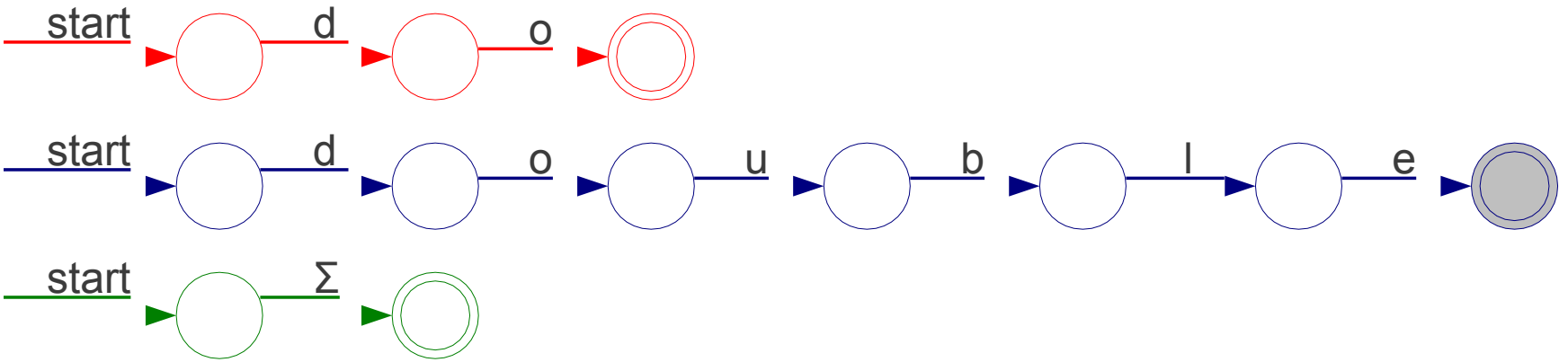
Implementing Maximal Munch

```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```



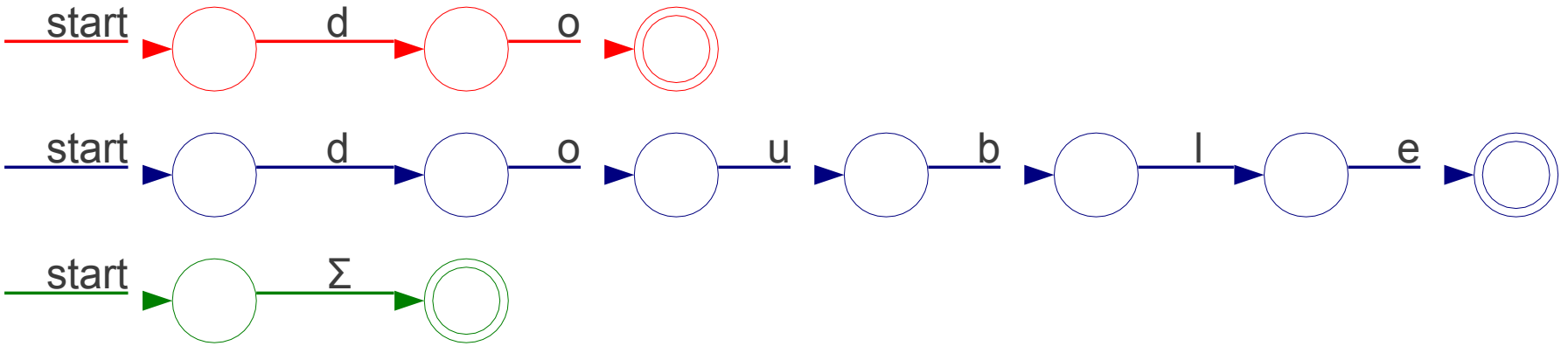
Implementing Maximal Munch

```
T_Do            do
T_Double        double
T_Mystery       [A-Za-z]
```

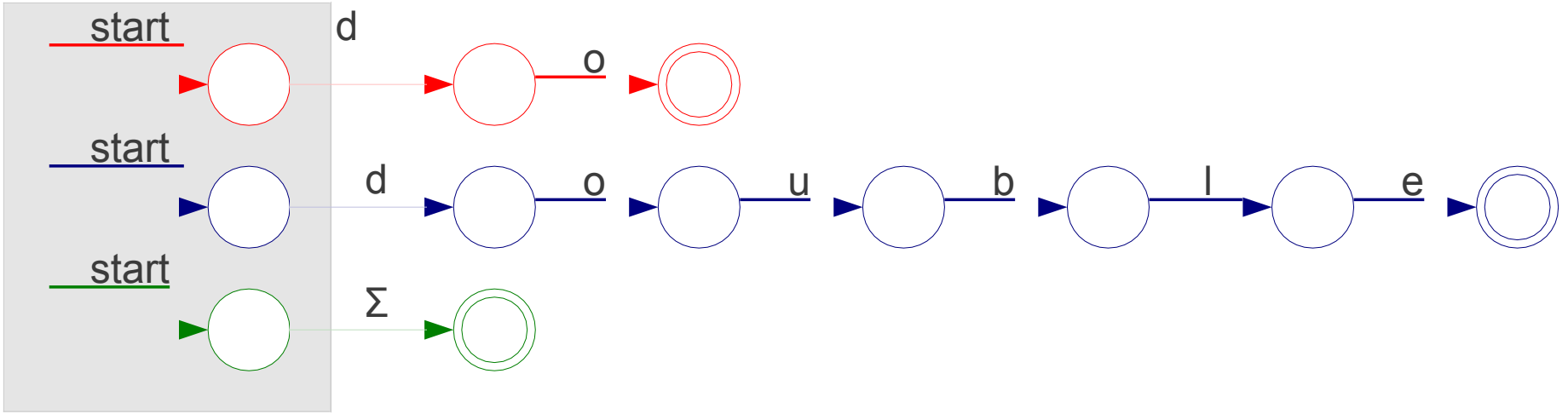


A Minor Simplification

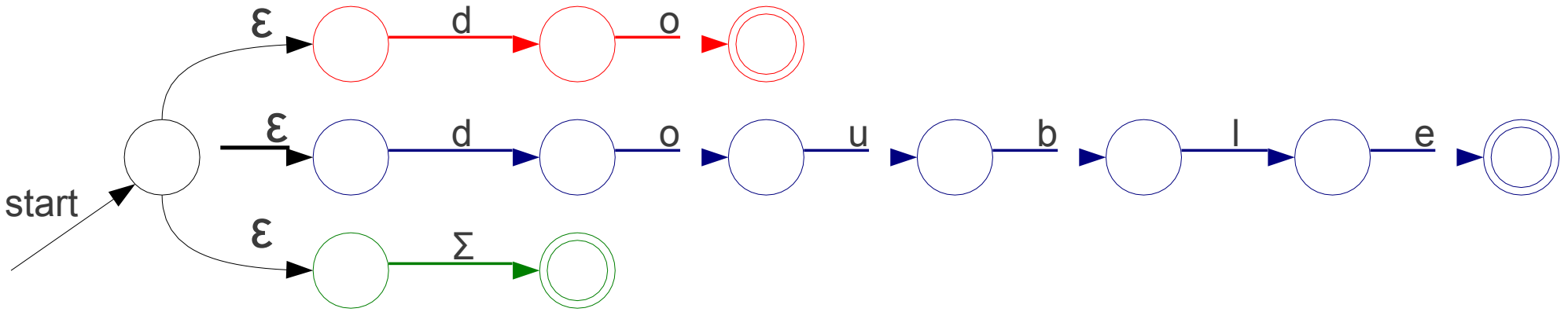
A Minor Simplification



A Minor Simplification

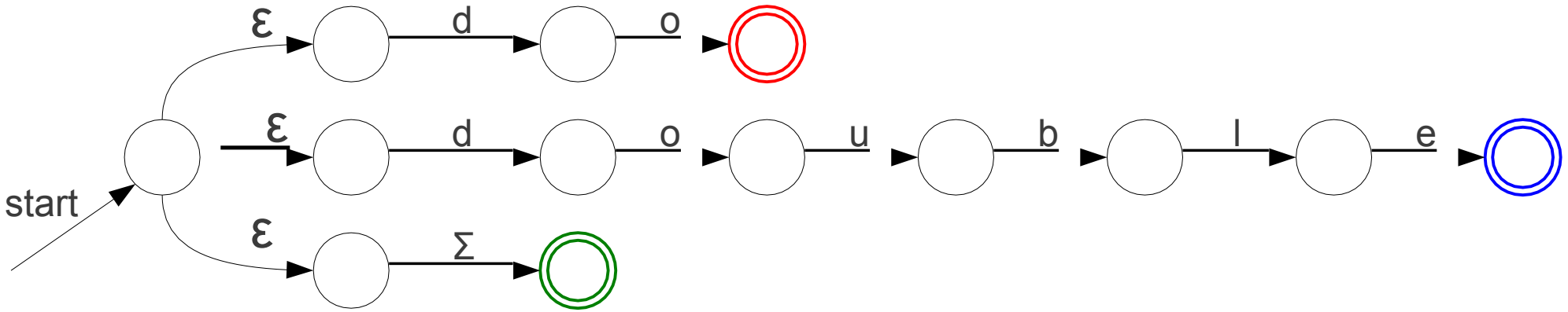


A Minor Simplification



Build a single automaton that runs all the matching automata in parallel.

A Minor Simplification



Annotate each accepting state with which automaton it came from.

Other Conflicts

```
T_Do          do
T_Double      double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```

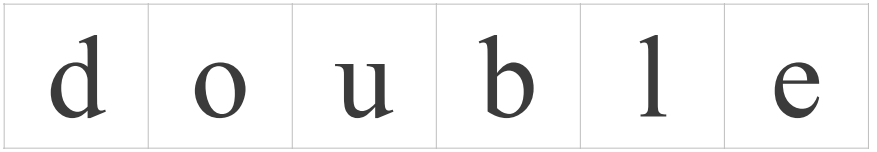
Other Conflicts

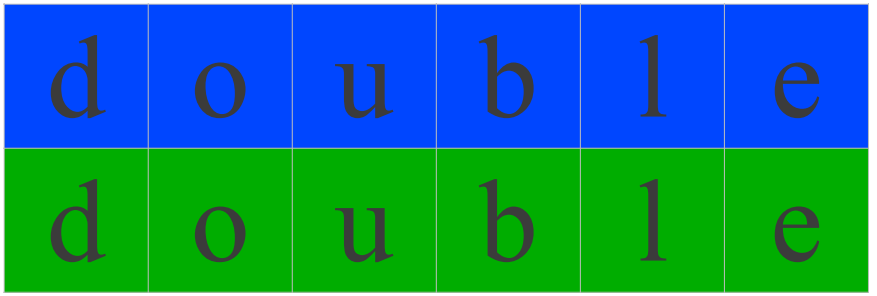
```
T_Do           do
T_Double       double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```

d	o	u	b	l	e

Other Conflicts

```
T_Do            do
T_Double        double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```



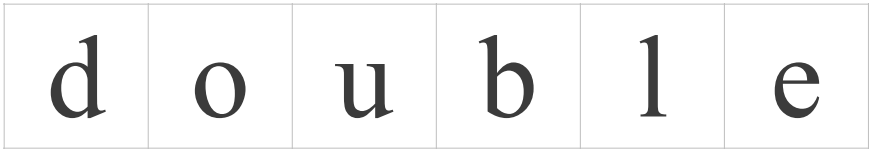


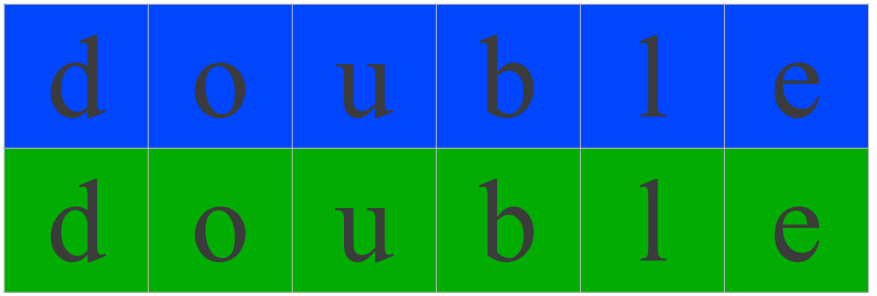
More Tiebreaking

- When two regular expressions apply, choose the one with the greater "priority."
- Simple priority system: **pick the rule that was defined first.**

Other Conflicts

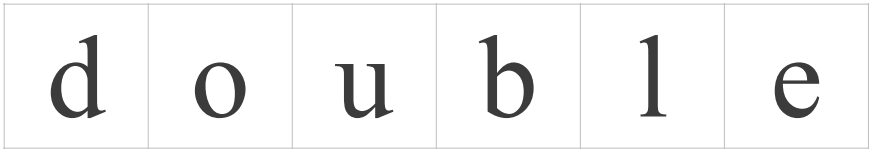
```
T_Do            do
T_Double        double
T_Identifier    [A-Za-z_][A-Za-z0-9_]*
```

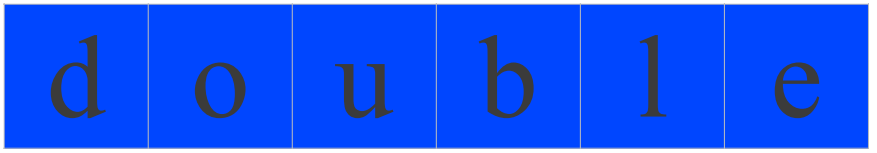




Other Conflicts

```
T_Do            do
T_Double        double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```





One Last Detail...

- We know what to do if *multiple* rules match.
- What if *nothing* matches?
- Trick: Add a “catch-all” rule that matches any character and reports an error.

Summary of Conflict Resolution

Summary of Conflict Resolution

- Construct an automaton for each regular expression.

Summary of Conflict Resolution

- Construct an automaton for each regular expression.
- Merge them into one automaton by adding a new start state.

Summary of Conflict Resolution

- Construct an automaton for each regular expression.
- Merge them into one automaton by adding a new start state.
- Scan the input, keeping track of the last known match.

Summary of Conflict Resolution

- Construct an automaton for each regular expression.
- Merge them into one automaton by adding a new start state.
- Scan the input, keeping track of the last known match.
- Break ties by choosing higher precedence matches.

Summary of Conflict Resolution

- Construct an automaton for each regular expression.
- Merge them into one automaton by adding a new start state.
- Scan the input, keeping track of the last known match.
- Break ties by choosing higher precedence matches.
- Have a catch-all rule to handle errors.

Challenges in Scanning

- How do we determine which lexemes are associated with each token?
- When there are multiple ways we could scan the input, how do we know which one to pick?
- How do we address these concerns efficiently?

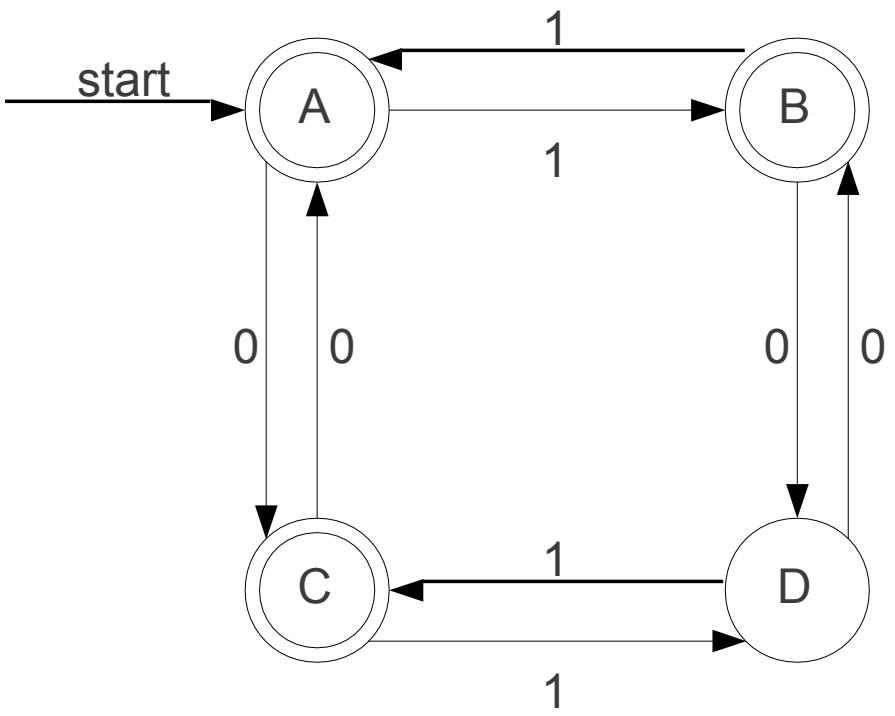
Challenges in Scanning

- How do we determine which lexemes are associated with each token?
- When there are multiple ways we could scan the input, how do we know which one to pick?
- How do we address these concerns efficiently?

DFAs

- The automata we've seen so far have all been NFAs.
- A **DFA** is like an NFA, but with tighter restrictions:
  - Every state must have **exactly one** transition defined for every letter.
  - ε-moves are not allowed.

A Sample DFA



	0	1
A.	C	B
B.	D	A
C.	A	D
D.	B	C

Speeding up Matching

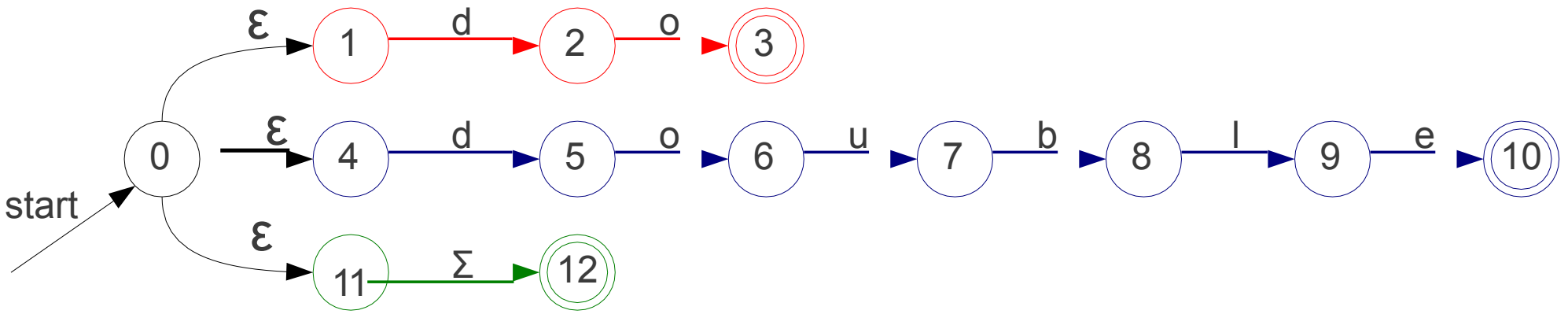
- In the worst-case, an NFA with n states takes time $O(mn^2)$ to match a string of length m.
- DFAs, on the other hand, take only $O(m)$.
- There is another (beautiful!) algorithm to convert NFAs to DFAs.

Subset Construction

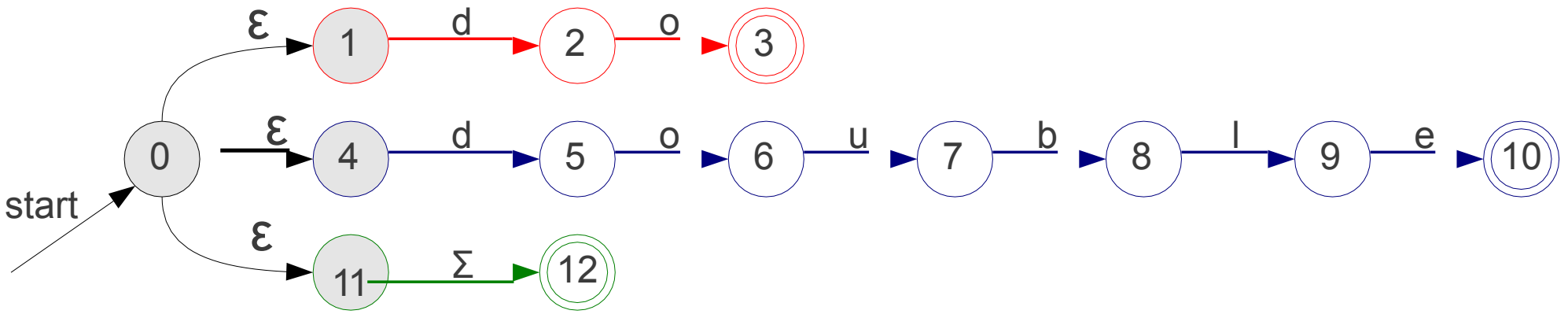
- NFAs can be in many states at once, while DFAs can only be in a single state at a time.
- Key idea: **Make the DFA simulate the NFA**.
- Have the states of the DFA correspond to the *sets of states* of the NFA.
- Transitions between states of DFA correspond to transitions between *sets of states* in the NFA.

From NFA to DFA

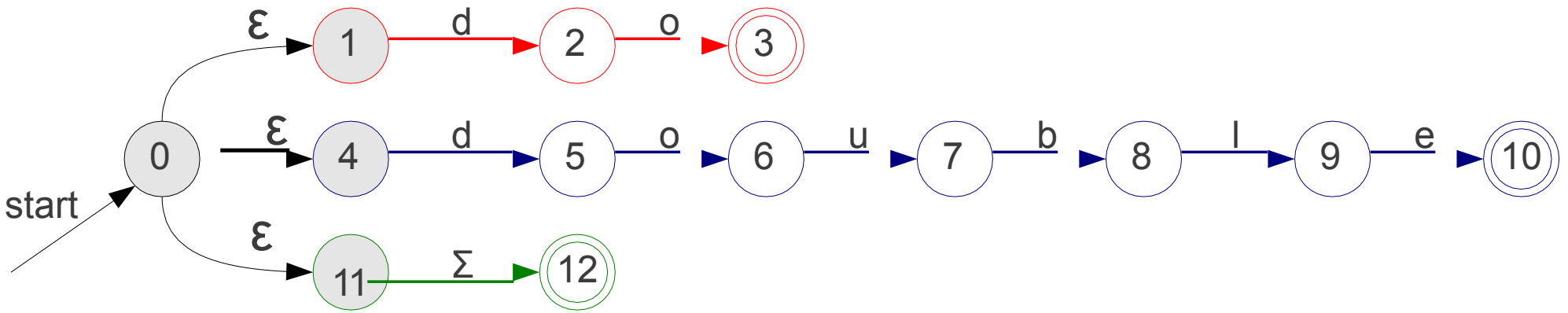
From NFA to DFA

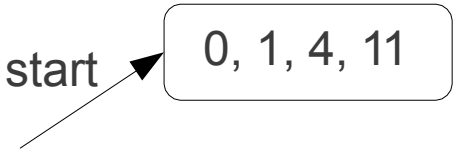


From NFA to DFA

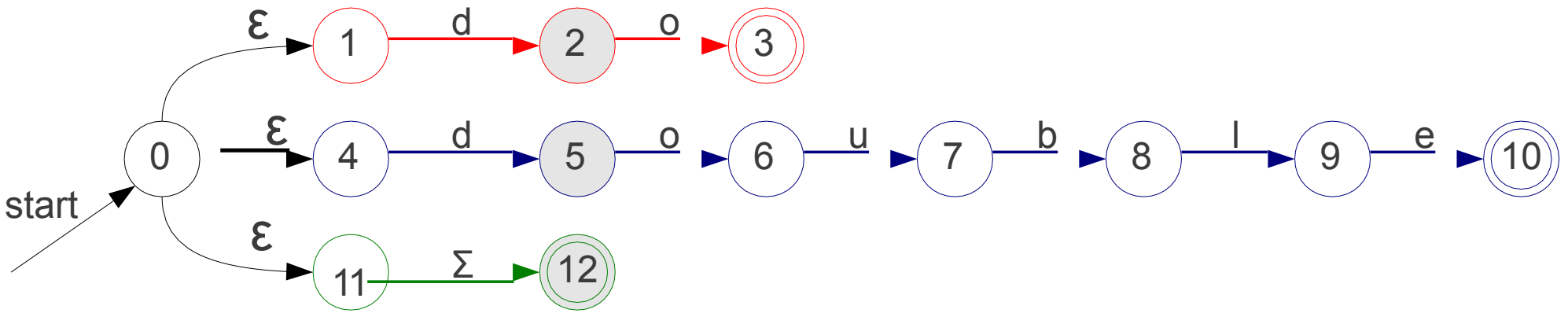


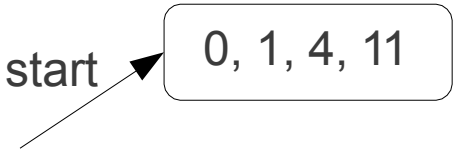
From NFA to DFA



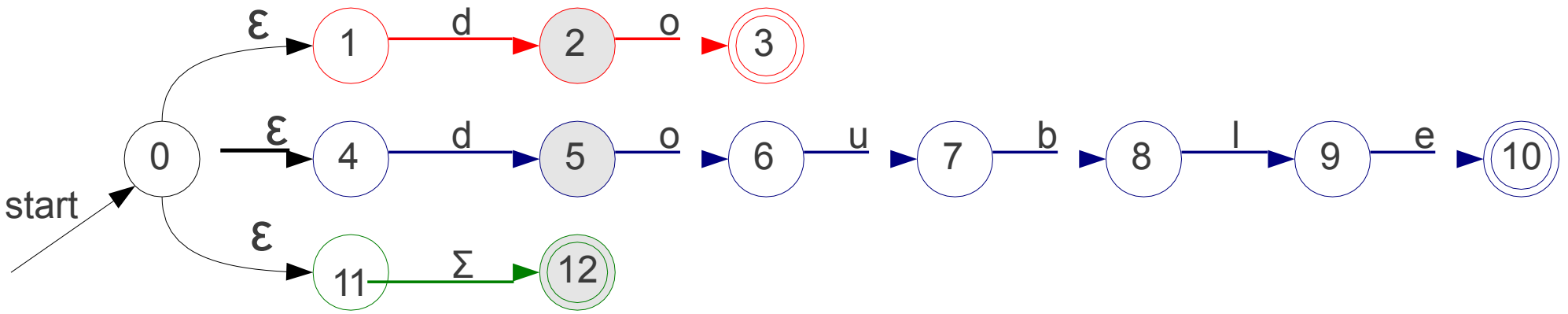


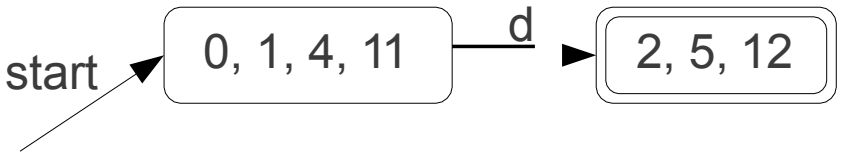
From NFA to DFA



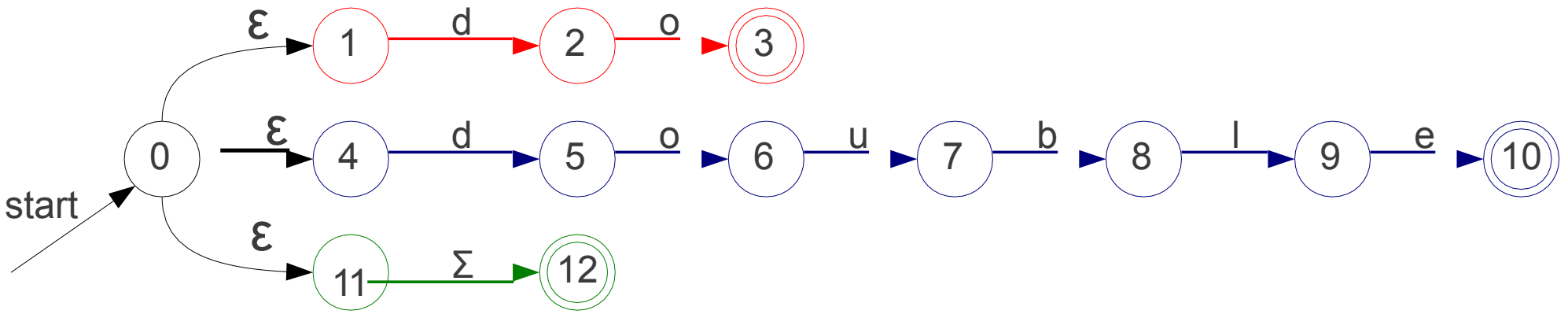


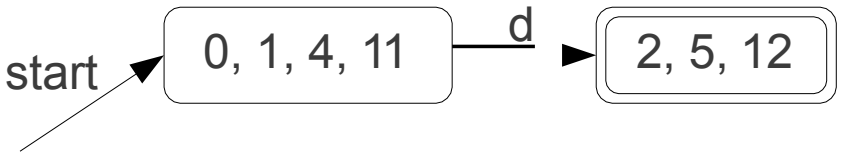
From NFA to DFA



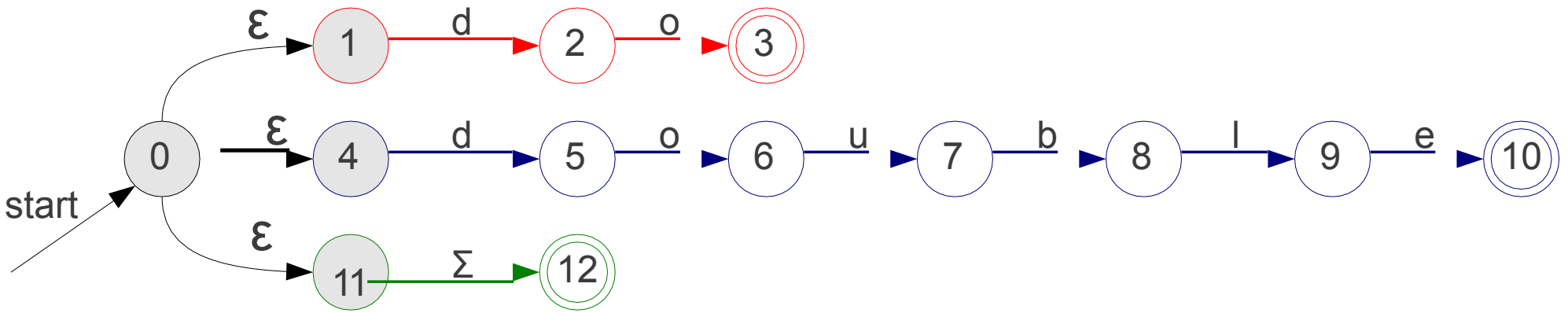


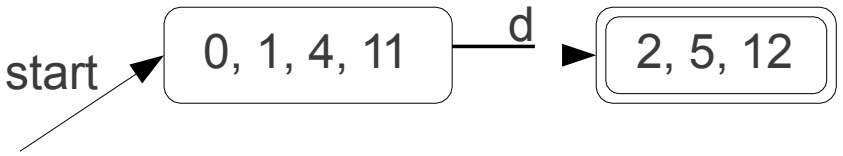
From NFA to DFA



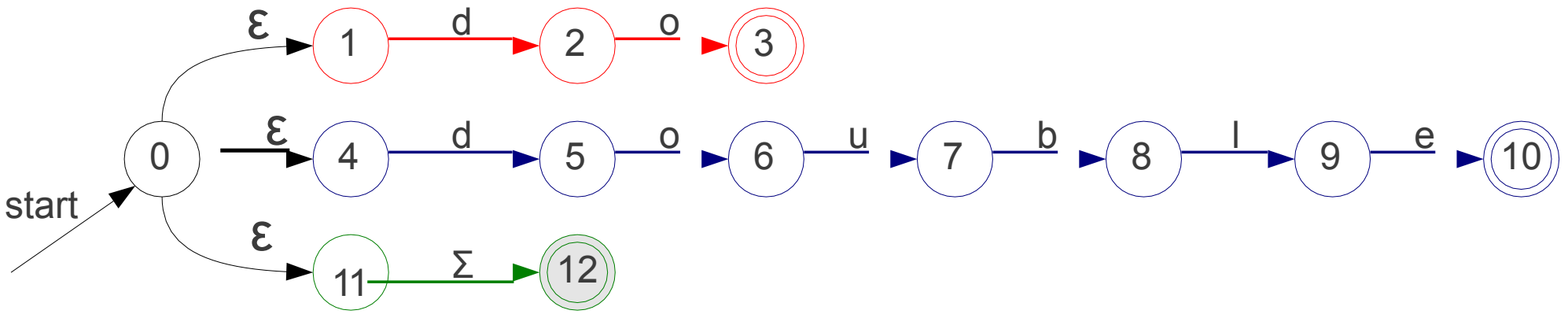


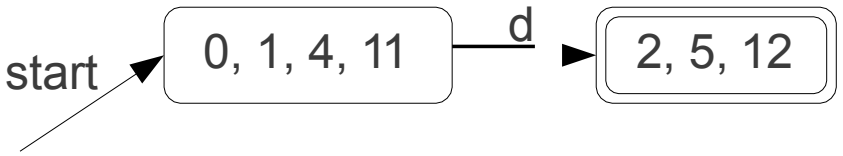
From NFA to DFA



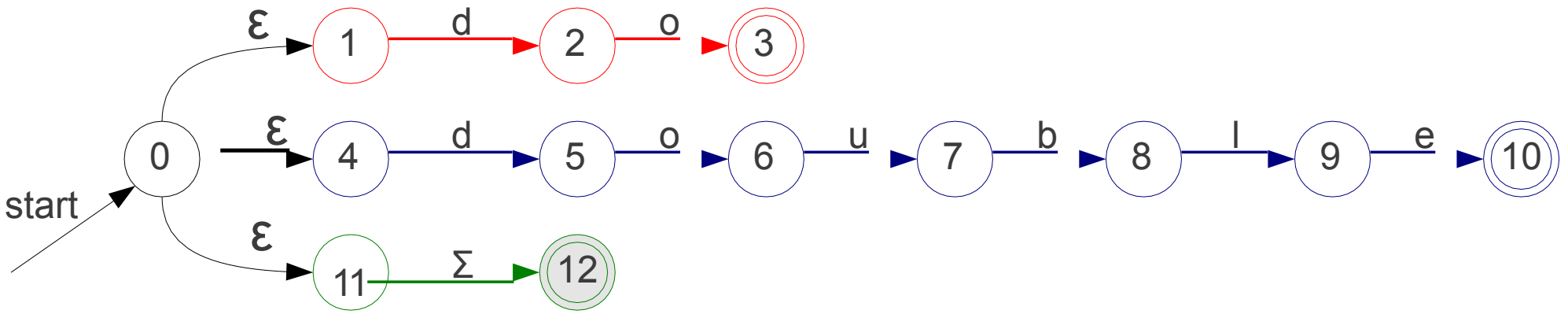


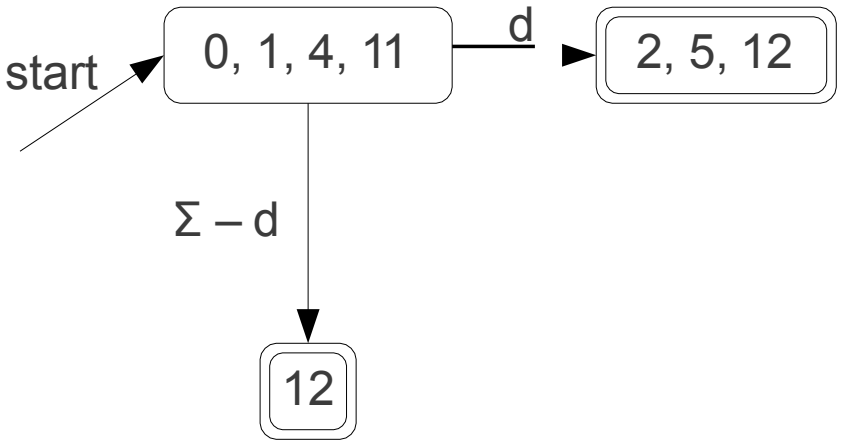
From NFA to DFA



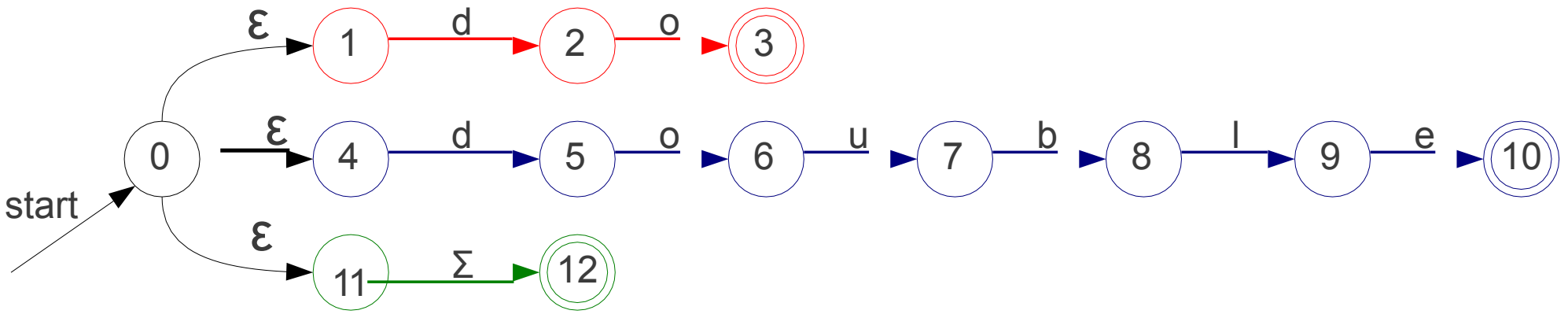


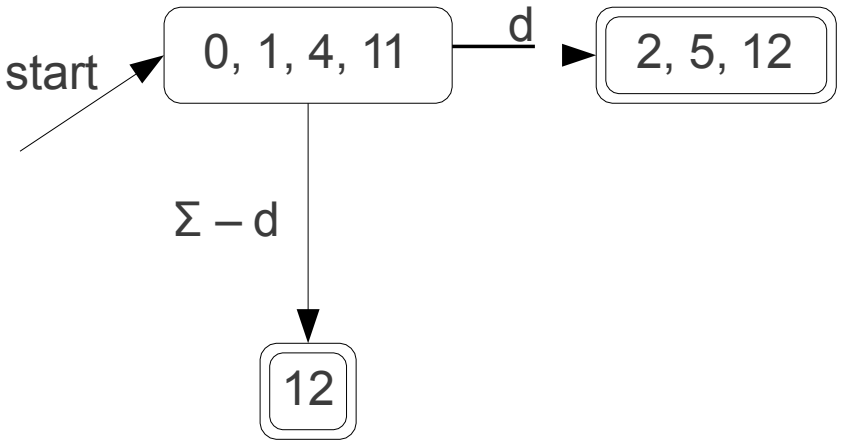
From NFA to DFA



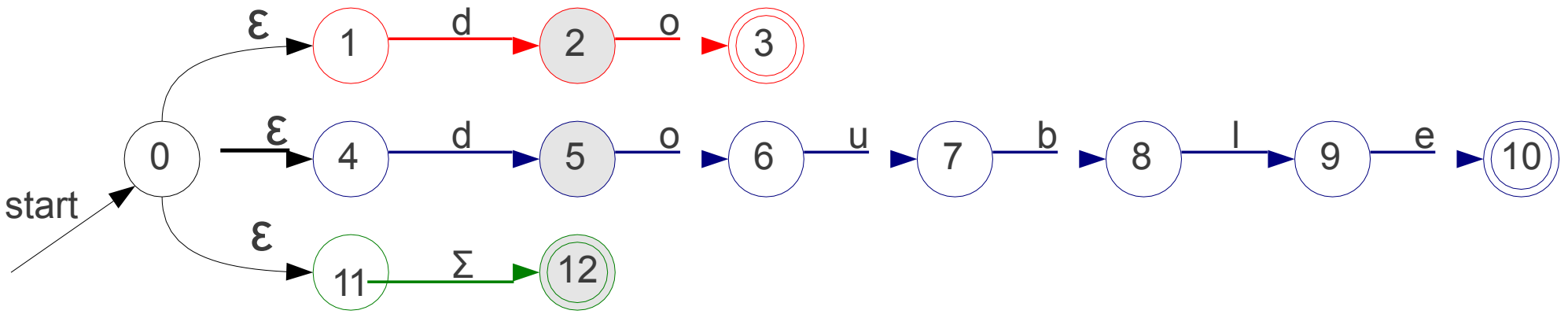


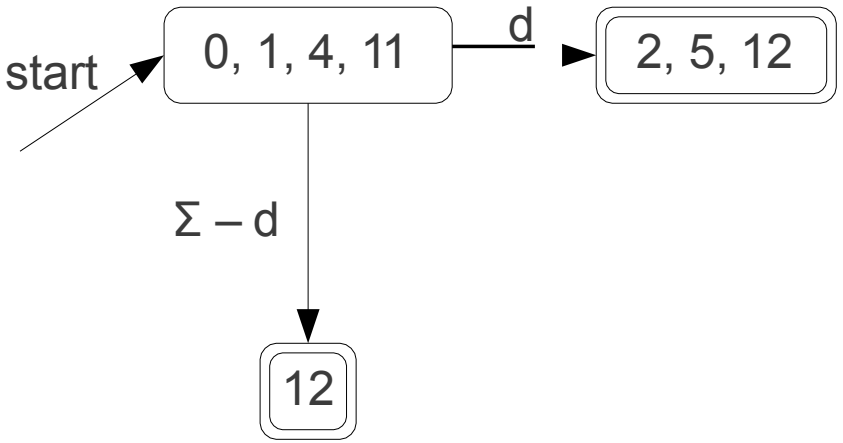
From NFA to DFA



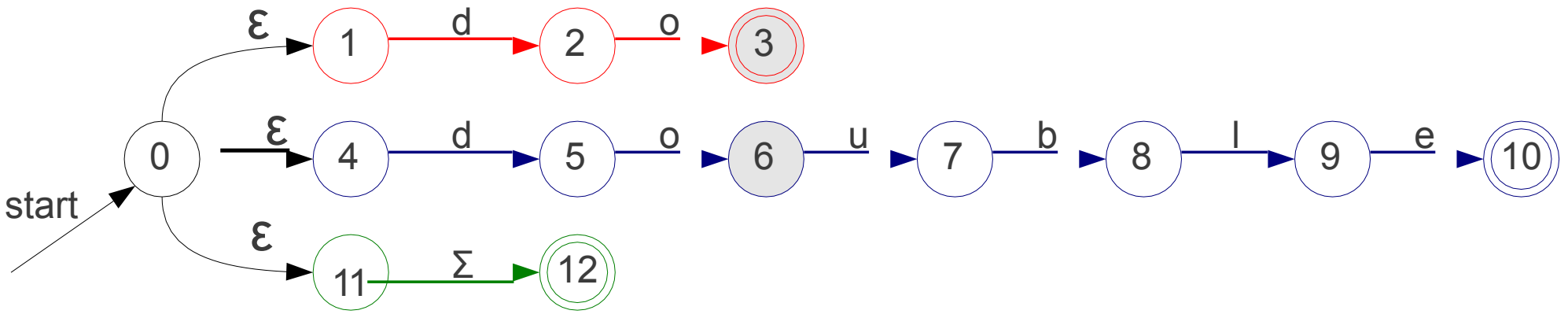


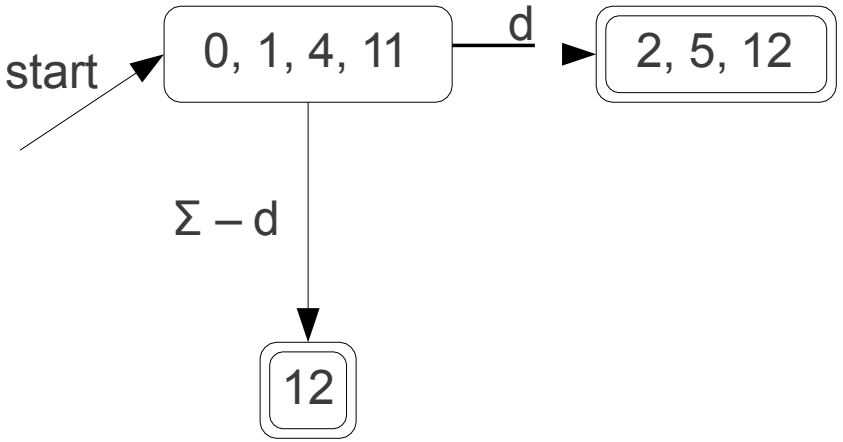
From NFA to DFA



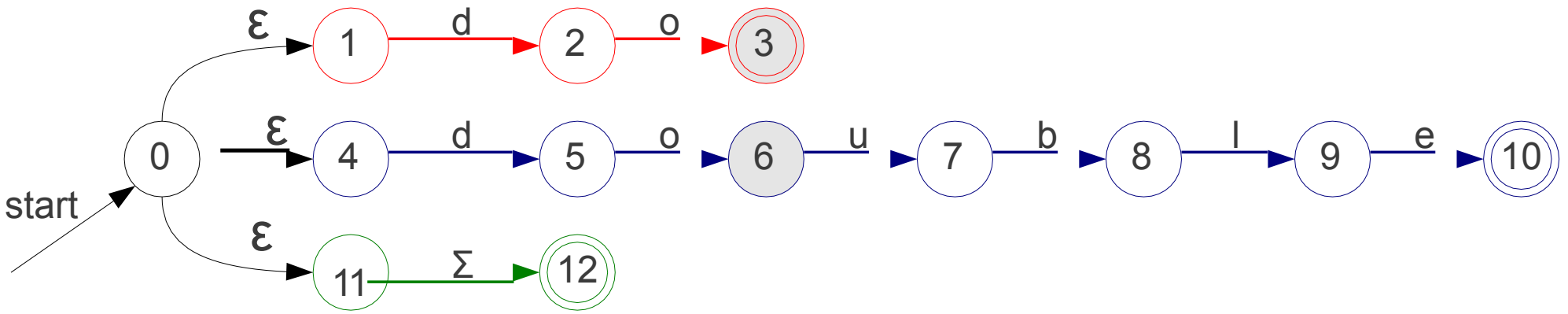


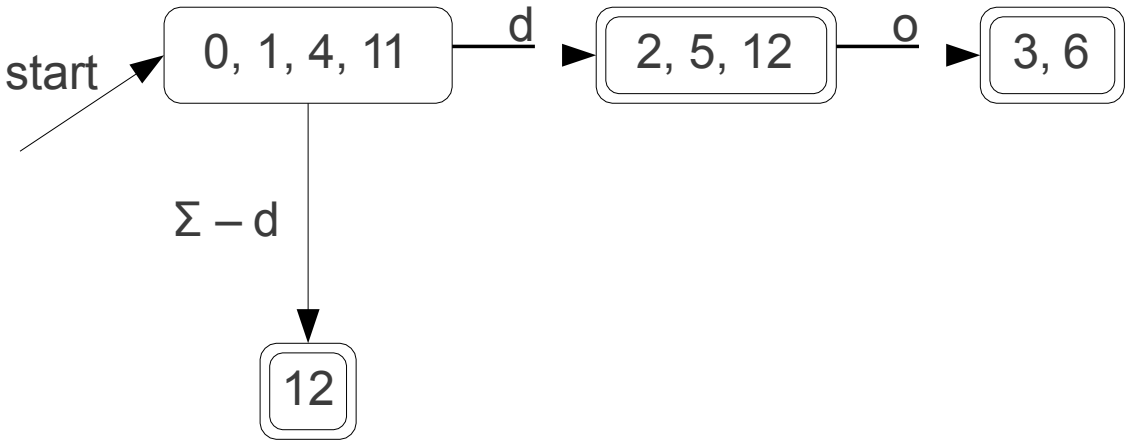
From NFA to DFA



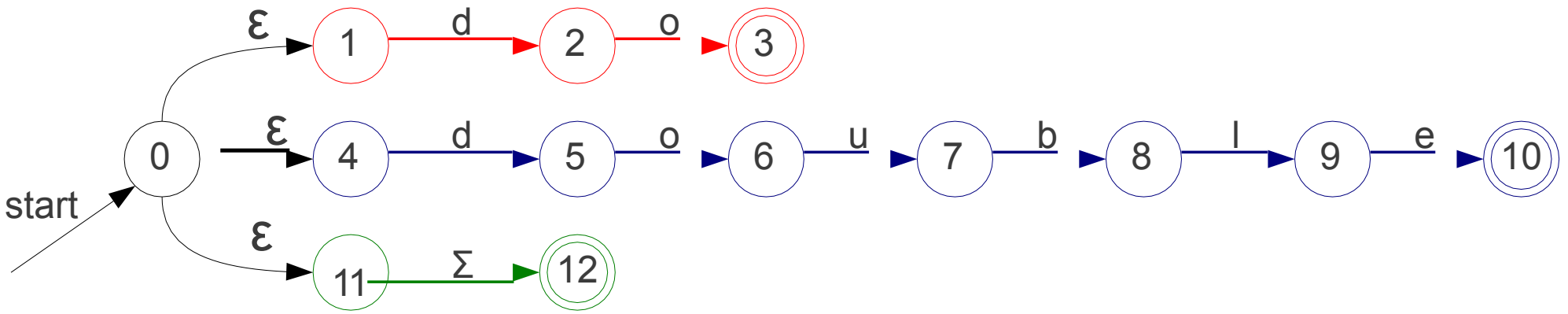


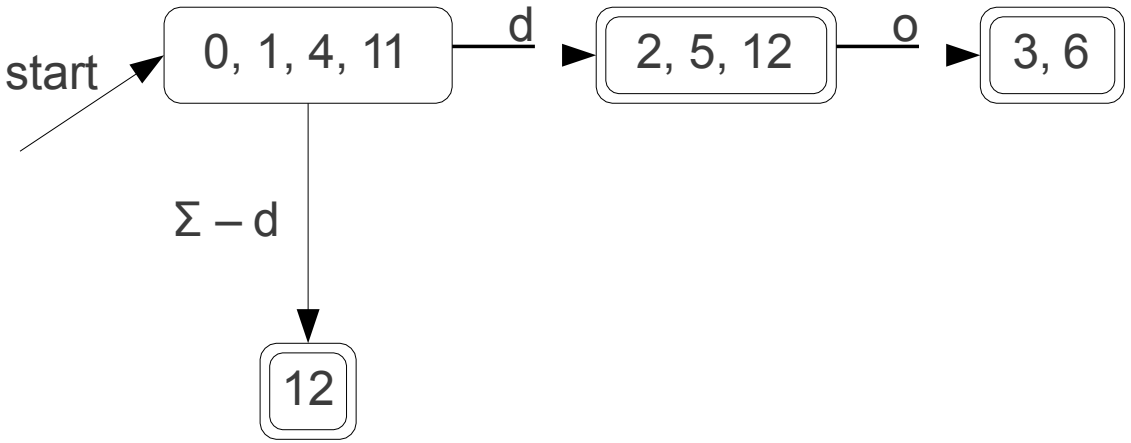
From NFA to DFA



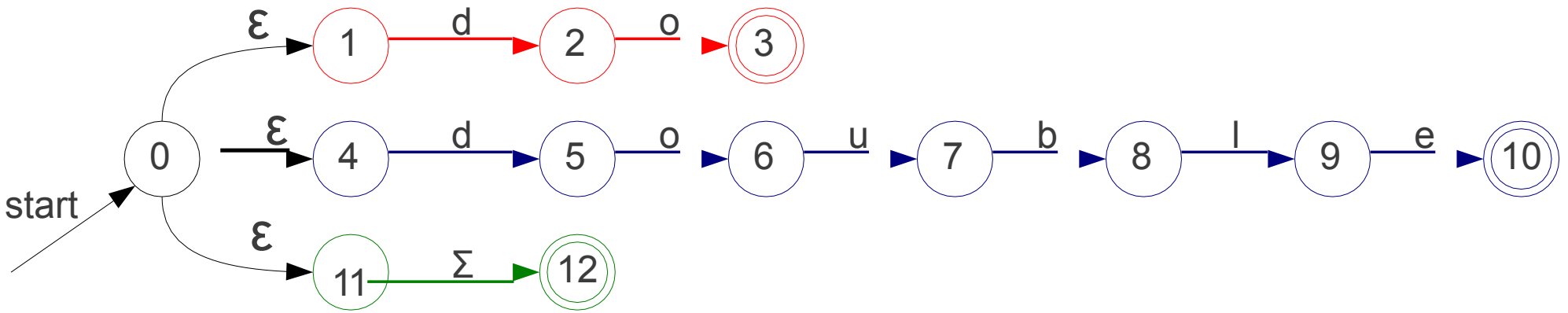


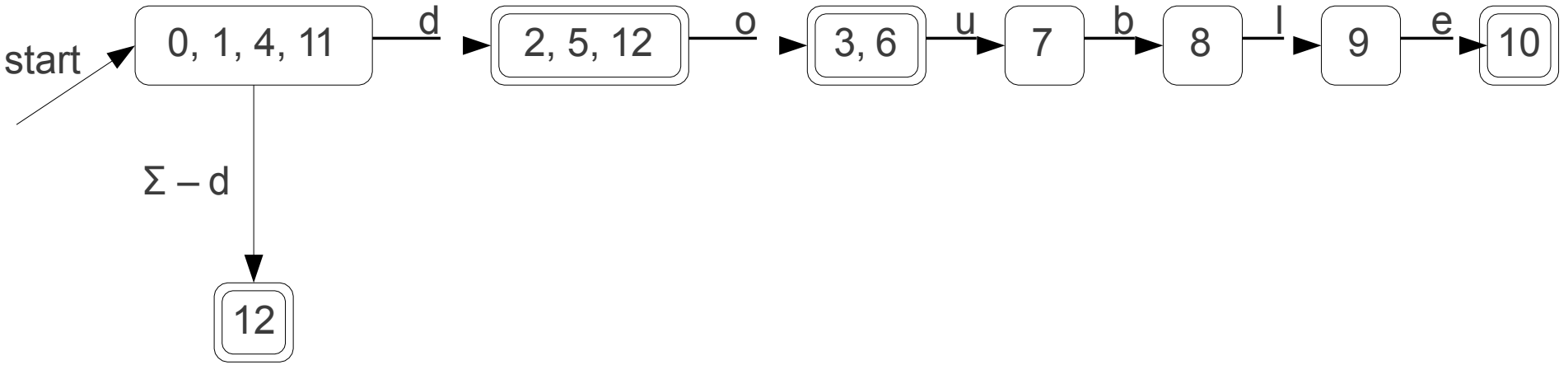
From NFA to DFA



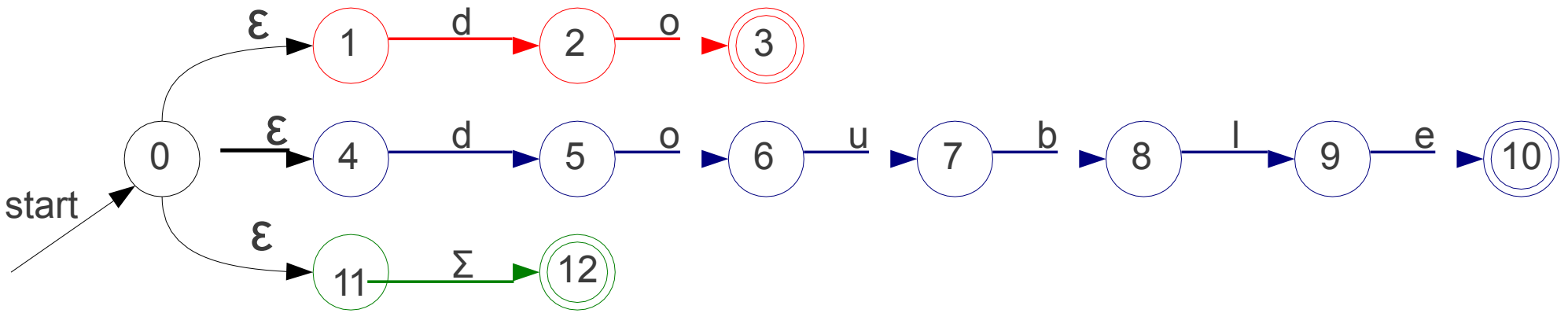


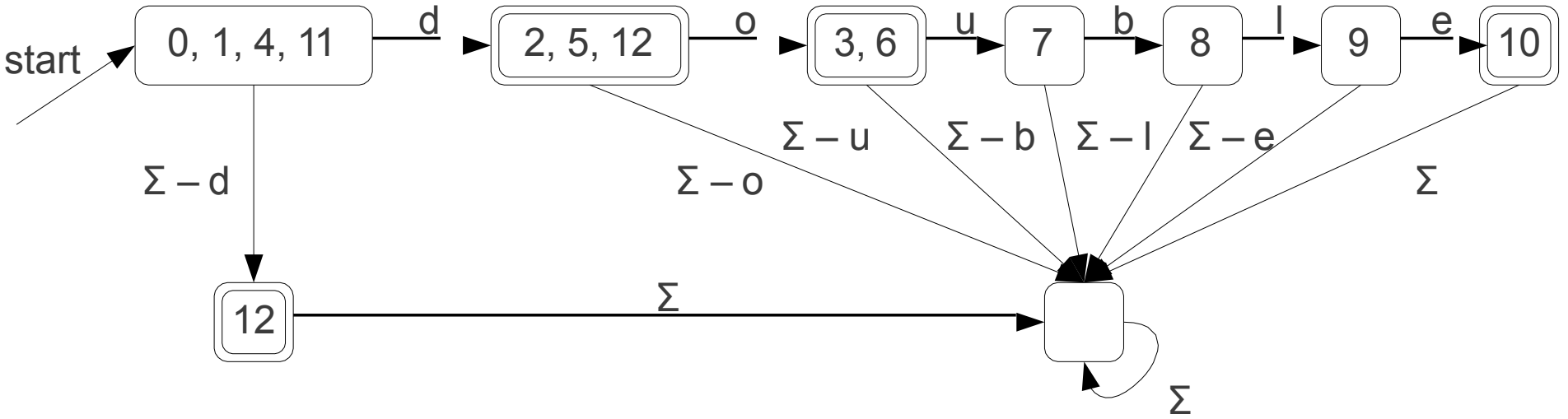
From NFA to DFA



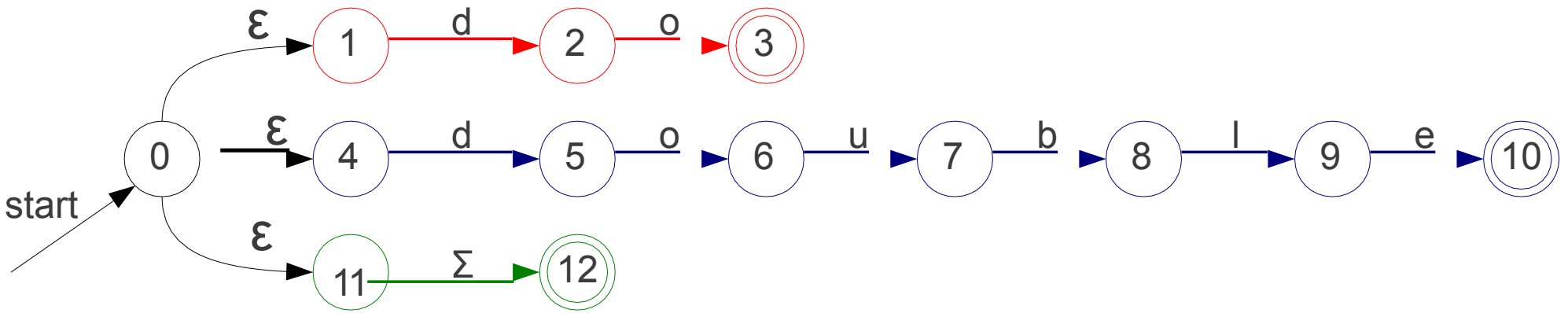


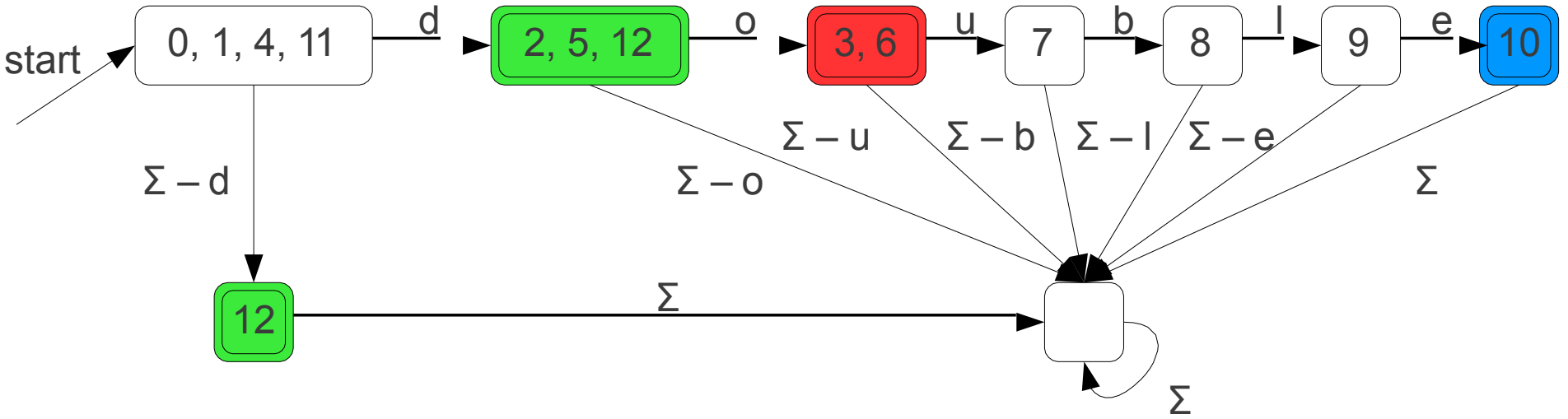
From NFA to DFA





From NFA to DFA

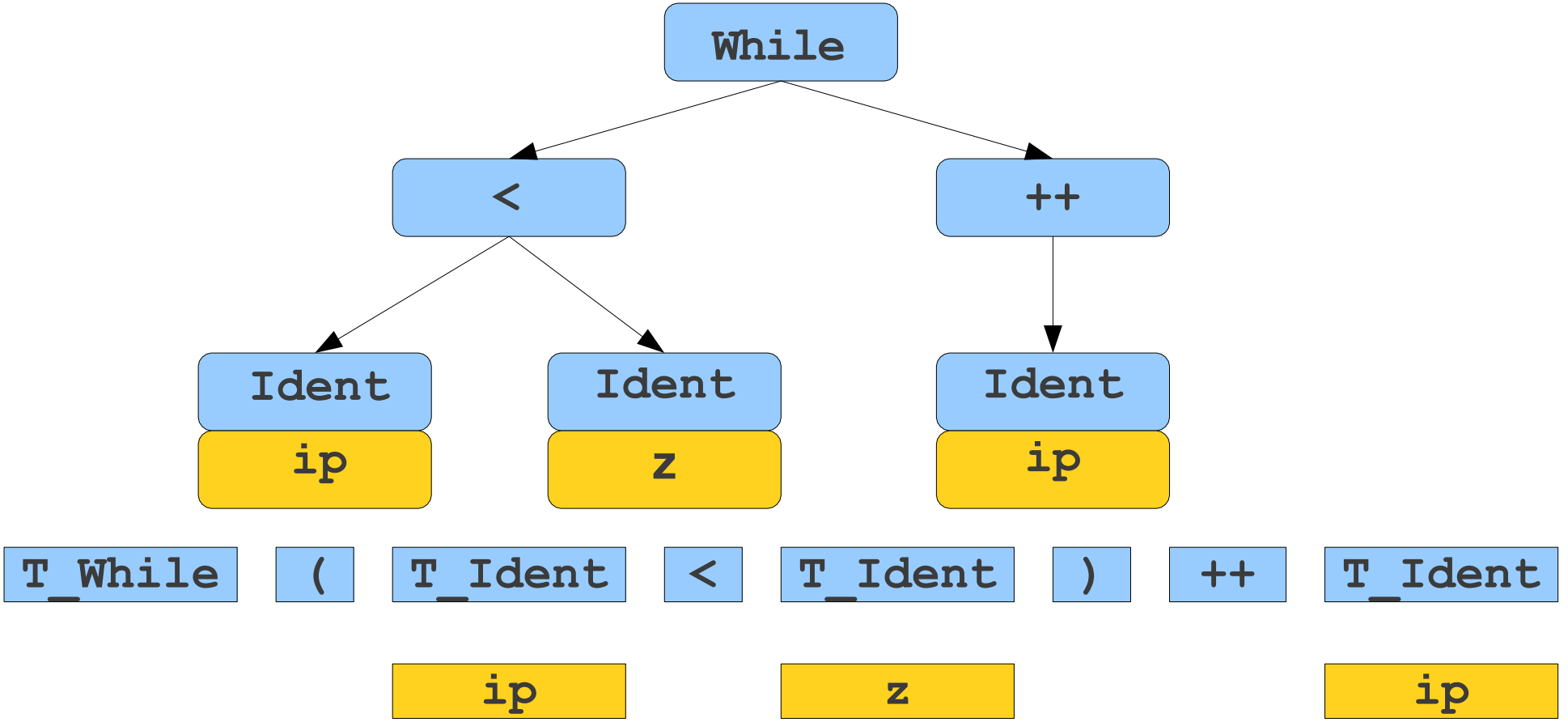




Performance Concerns

- The NFA-to-DFA construction can introduce *exponentially* many states.
- Time/memory tradeoff:
  - Low-memory NFA has higher scan time.
  - High-memory DFA has lower scan time.
- Could use a hybrid approach by simplifying NFA before generating code.

Real-World Scanning: **Python**



w	h	i	l	e		(	i	p		<		z	)	\n\t	+	+	i	p	;

```
while (ip < z)
    ++ip;
```

Python Blocks

- Scoping handled by whitespace:

```
if w == z:
    a = b
    c = d
else
    e = f
g =
  h
```

- What does that mean for the scanner?

Whitespace Tokens

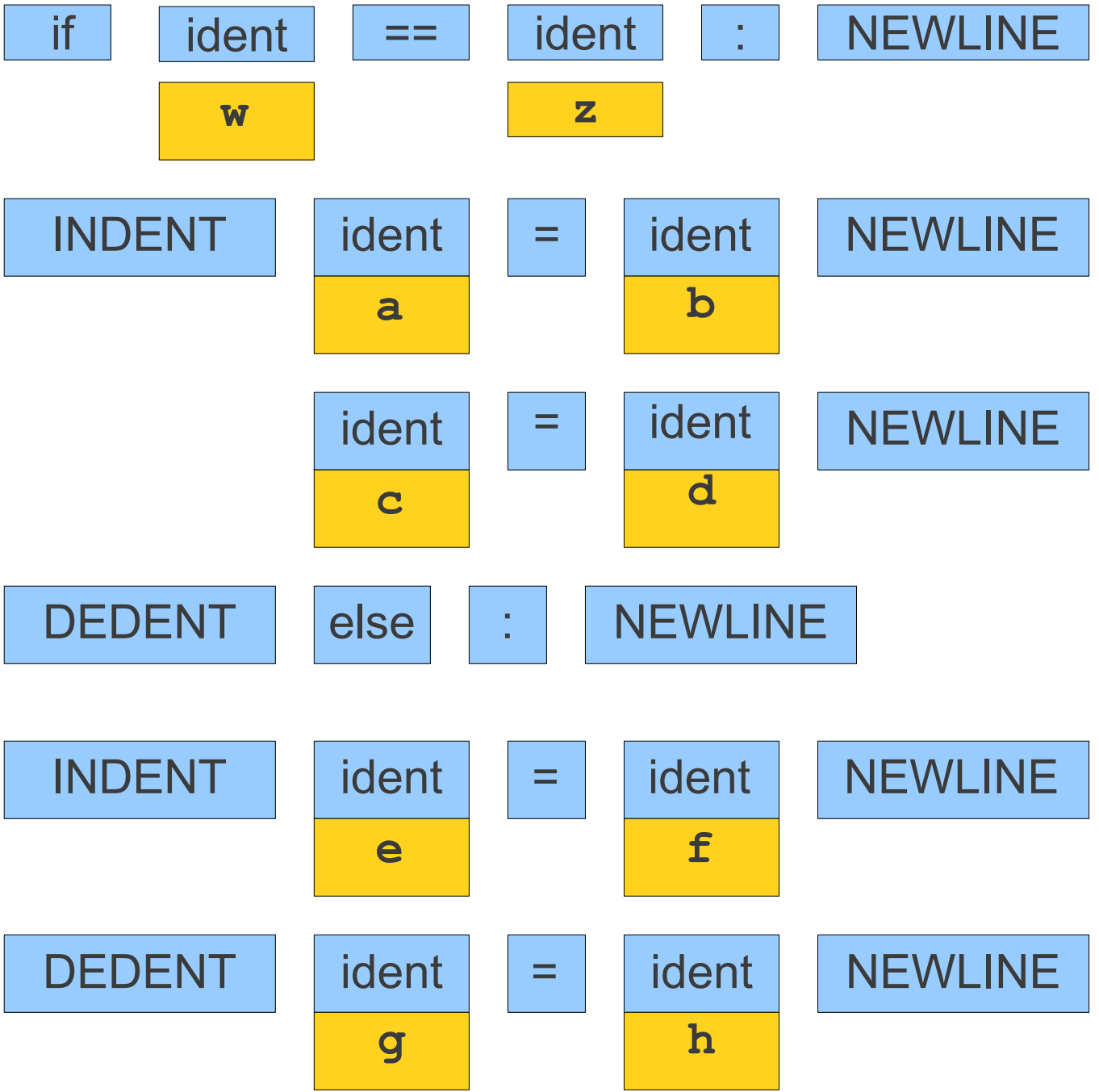
- Special tokens inserted to indicate changes in levels of indentation.
- **NEWLINE** marks the end of a line.
- **INDENT** indicates an increase in indentation.
- **DEDENT** indicates a decrease in indentation. Note
- that INDENT and DEDENT encode *change* in indentation, not the total amount of indentation.

Scanning Python

```
if w == z:
    a = b
    c = d
 else
    :
    e = f
  g =
    h
```

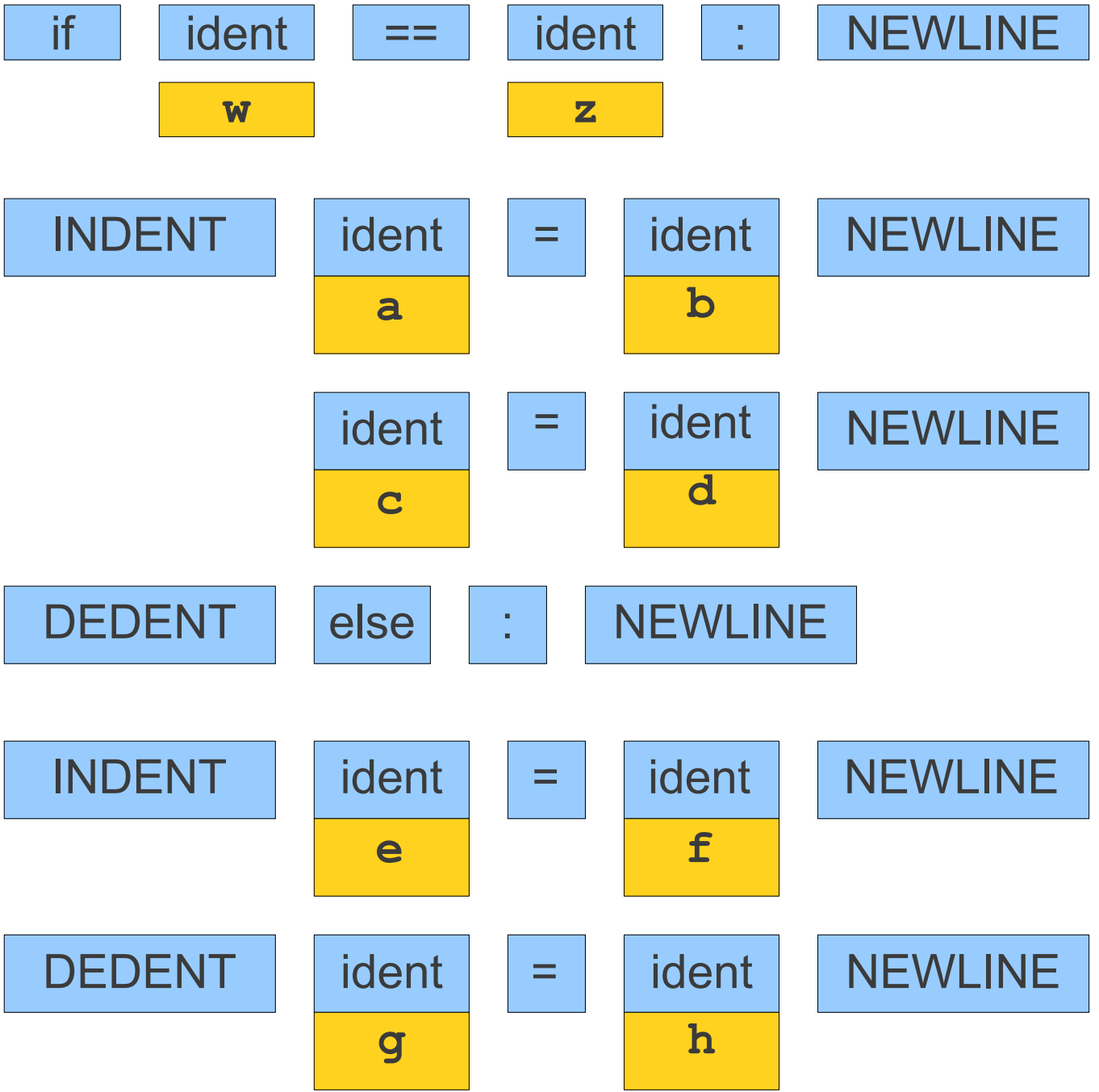
Scanning Python

```
if w == z:
    a = b
    c = d
else:
    e = f
g = h
```



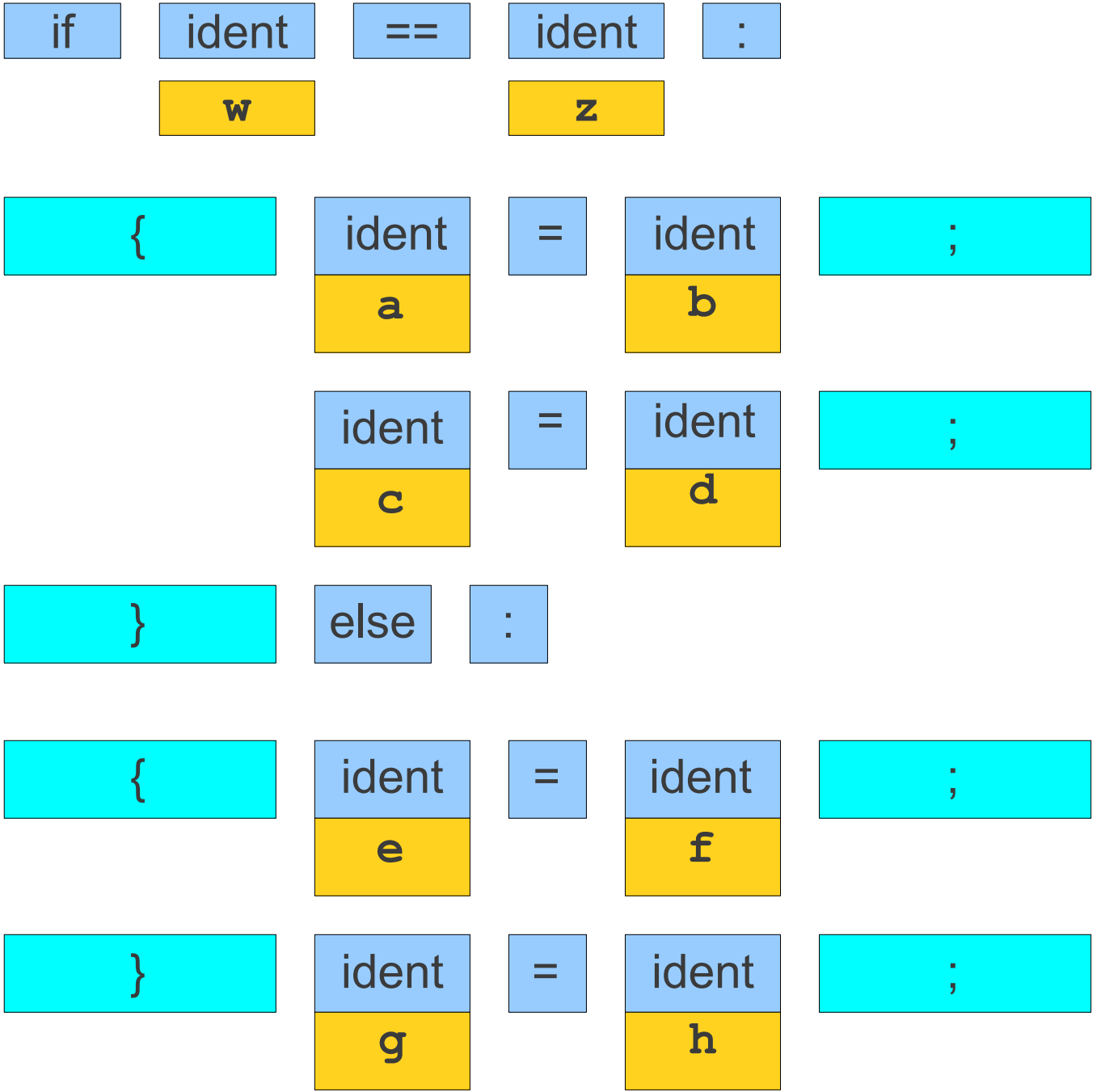
Scanning Python

```
if w == z: {
    a = b;
    c = d;
} else {
    e = f;
}
g = h;
```



Scanning Python

```
if w == z: {
    a = b;
    c = d;
} else {
    e = f;
}
g = h;
```



Where to INDENT/DEDENT?

-
-
  -
  -

  Otherwise, while the value is less than the top of the stack:

Source: http://docs.python.org/reference/lexical_analysis.html

Where to INDENT/DEDENT?

Scanner maintains a stack of line indentations keeping track of all indented contexts so far.

-
-
  -
  -

  Otherwise, while the value is less than the top of the stack:

Source: http://docs.python.org/reference/lexical_analysis.html

Where to INDENT/DEDENT?

Scanner maintains a stack of line indentations keeping track of all indented contexts so far.

Initially, this stack contains 0, since initially the contents of the file aren't indented.

-
-
  -
  -

  Otherwise, while the value is less than the top of the stack:

Source: http://docs.python.org/reference/lexical_analysis.html

Where to INDENT/DEDENT?

Scanner maintains a stack of line indentations keeping track of all indented contexts so far.

Initially, this stack contains 0, since initially the contents of the file aren't indented.

On a newline:

-
-
  -
  -

  Otherwise, while the value is less than the top of the stack:

Source: http://docs.python.org/reference/lexical_analysis.html

Where to INDENT/DEDENT?

Scanner maintains a stack of line indentations keeping track of all indented contexts so far.

Initially, this stack contains 0, since initially the contents of the file aren't indented.

On a newline:

- See how much whitespace is at the start of the line. If this
- value exceeds the top of the stack:
  -
  -

  Otherwise, while the value is less than the top of the stack:

Source: http://docs.python.org/reference/lexical_analysis.html

Where to INDENT/DEDENT?

Scanner maintains a stack of line indentations keeping track of all indented contexts so far.

Initially, this stack contains 0, since initially the contents of the file aren't indented.

On a newline:

- See how much whitespace is at the start of the line. If this
- value exceeds the top of the stack:
  - Push the value onto the stack.
  - Emit an INDENT token.

  Otherwise, while the value is less than the top of the stack:

Source: http://docs.python.org/reference/lexical_analysis.html

Where to INDENT/DEDENT?

Scanner maintains a stack of line indentations keeping track of all indented contexts so far.

Initially, this stack contains 0, since initially the contents of the file aren't indented.

On a newline:

- See how much whitespace is at the start of the line. If this
- value exceeds the top of the stack:
  - Push the value onto the stack.
  - Emit an INDENT token.
- Otherwise, while the value is less than the top of the stack:
  - Pop the stack.
  - Emit a DEDENT token.

Source: http://docs.python.org/reference/lexical_analysis.html

Summary

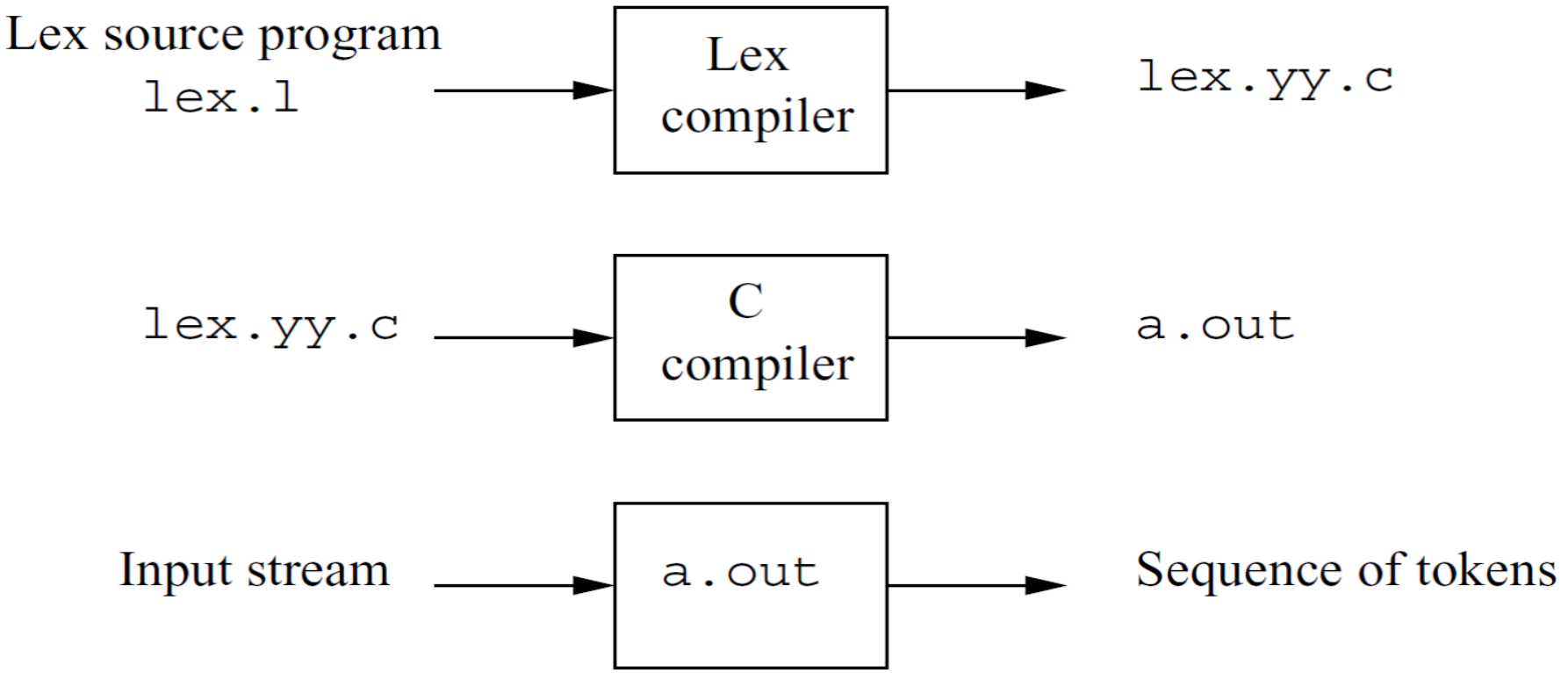
- Lexical analysis splits input text into **tokens** holding a **lexeme** and an **attribute**.
- Lexemes are sets of strings often defined with **regular expressions**.
- Regular expressions can be converted to **NFAs** and from there to **DFAs**.
- **Maximal-munch** using an automaton allows for fast scanning.
- Not all tokens come directly from the source code.

Introduction To `flex`

What is flex?

- Automated tool for generating scanners.
- Uses maximal-munch/precedence system.
- Internally, builds a DFA from regular expressions.

(F) `lex` Structure



Structure of flex file

```
declarations
%%
translation rules
%%
auxiliary functions
```

The **declarations** section includes declarations of variables, manifest constants

(identifiers declared to stand for a constant, e.g., the name of a **token**), and regular dentitions.

Structure of flex file

declarations
%%
translation rules
%%
auxiliary functions

The **translation rules** each have the form:

```
Pattern {Action}
```

Each pattern is a regular expression, which may use the regular definitions of the declaration section.

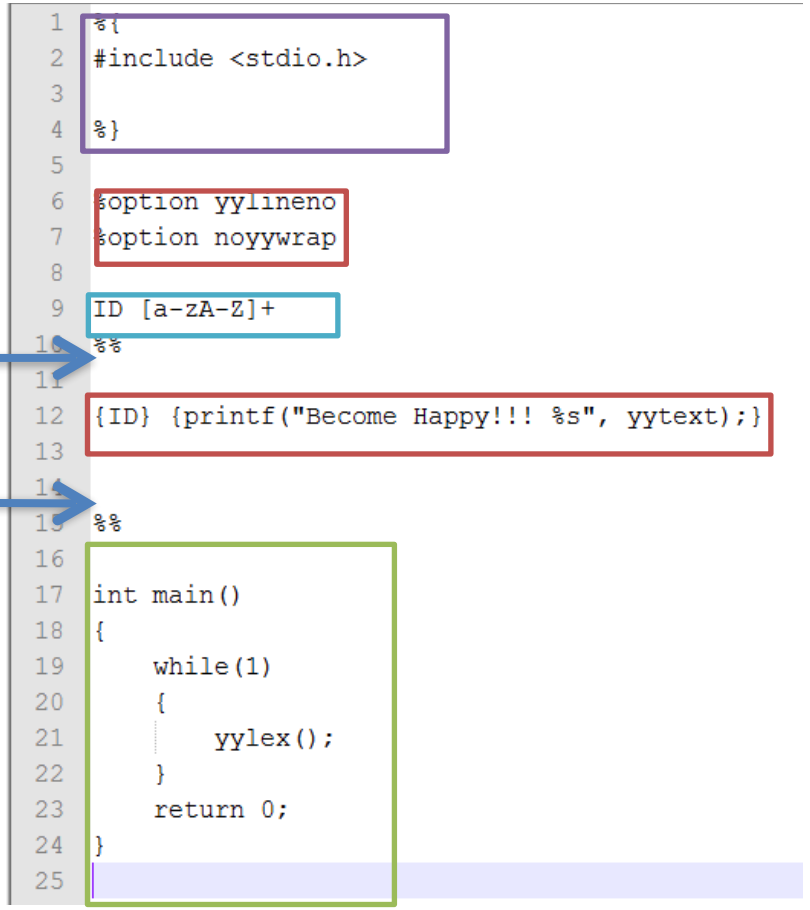
declarations

%%

translation rules

%%

auxiliary functions



```
%{
#include <stdio.h>

%}

%option yylineno
%option noyywrap

ID [a-zA-Z]+
%%

{ID} {printf("Become Happy!!! %s", yytext);}

%%

int main()
{
    while(1)
    {
        yylex();
    }
    return 0;
}
```

Work on an Example

- Imagine we have the following definitions:

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**number**	Pointer to table entry
`<`	**relop**	`LT`
`<=`	**relop**	`LE`
`=`	**relop**	`EQ`
`<>`	**relop**	`NE`
`>`	**relop**	`GT`
`>=`	**relop**	`GE`

Example: definition

```
%{
/* definitions of manifest constants
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP *
/
%}
/* regular definitions */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%
```

Example: definition

```
%{
/* definitions of manifest constants
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP *
/
%}
/* regular definitions */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%
```

[s] : any one of char in string *s*

Example: definition

```
%{
/* definitions of manifest constants
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP *
/
%}
/* regular definitions */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%
```

[s] : any one of char in string *s*

\- : implies range

Example: definition

```
%{
/* definitions of manifest constants
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP *
/
%}
/* regular definitions */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%
```

[s] : any one of char in string *s*

\- : implies range

***, +, ?**: zero or more, one or more, zero or one

Example: definition

```
%{
/* definitions of manifest constants
LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE, ID, NUMBER, RELOP *
/
%}
/* regular definitions */
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%
```

[s] : any one of char in string *s*

\- : implies range

***, +, ?**: zero or more, one or more, zero or one

Complete Guide can be found in Book figure 3.8

Example: Transition

```
{ws} {/* no action and no return */}
if     {return(IF);}
then   {return(THEN);}
else   {return(ELSE);}
{id}   {yylval = (int) installID(); return(ID);}
{number} {yylval = (int) installNum(); return(NUMBER);}
"<"  {yylval  = LT; return(RELOP);}
"<=" {yylval  = LE; return(RELOP);}
"="  {yylval  = EQ; return(RELOP);}
"<>" {yylval  = NE; return(RELOP);}
">"  {yylval  = GT; return(RELOP);}
">=" {yylval  = GE; return(RELOP);}
%%
```

Example: Transition

```
{ws} {/* no action and no return */}
if     {return(IF);}
then   {return(THEN);}
else   {return(ELSE);}
{id}   {yylval = (int) installID(); return(ID);}
{number} {yylval = (int) installNum(); return(NUMBER);}
"<"  {yylval  = LT; return(RELOP);}
"<=" {yylval  = LE; return(RELOP);}
"="  {yylval  = EQ; return(RELOP);}
"<>" {yylval  = NE; return(RELOP);}
">"  {yylval  = GT; return(RELOP);}
">=" {yylval  = GE; return(RELOP);}
%%
```

We can use C++ inside **{}**

Example: Transition

```
{ws} {/* no action and no return */}
if     {return(IF);}
then   {return(THEN);}
else   {return(ELSE);}
{id}   {yylval = (int) installID(); return(ID);}
{number} {yylval = (int) installNum(); return(NUMBER);}
"<"  {yylval  = LT; return(RELOP);}
"<=" {yylval  = LE; return(RELOP);}
"="  {yylval  = EQ; return(RELOP);}
"<>" {yylval  = NE; return(RELOP);}
">"  {yylval  = GT; return(RELOP);}
">=" {yylval  = GE; return(RELOP);}
%%
```

We can use C++ inside **{}**

`yytext()` contains lexemes.

Example: auxiliary functions

```
int installID() {/* function to install the lexeme, whose first
character is pointed to by yytext,
and whose length is yyleng, into the
symbol table and return a pointer thereto */}

int installNum() {/
* similar to installID, but puts numerical constants into a separ
ate table */
}
```

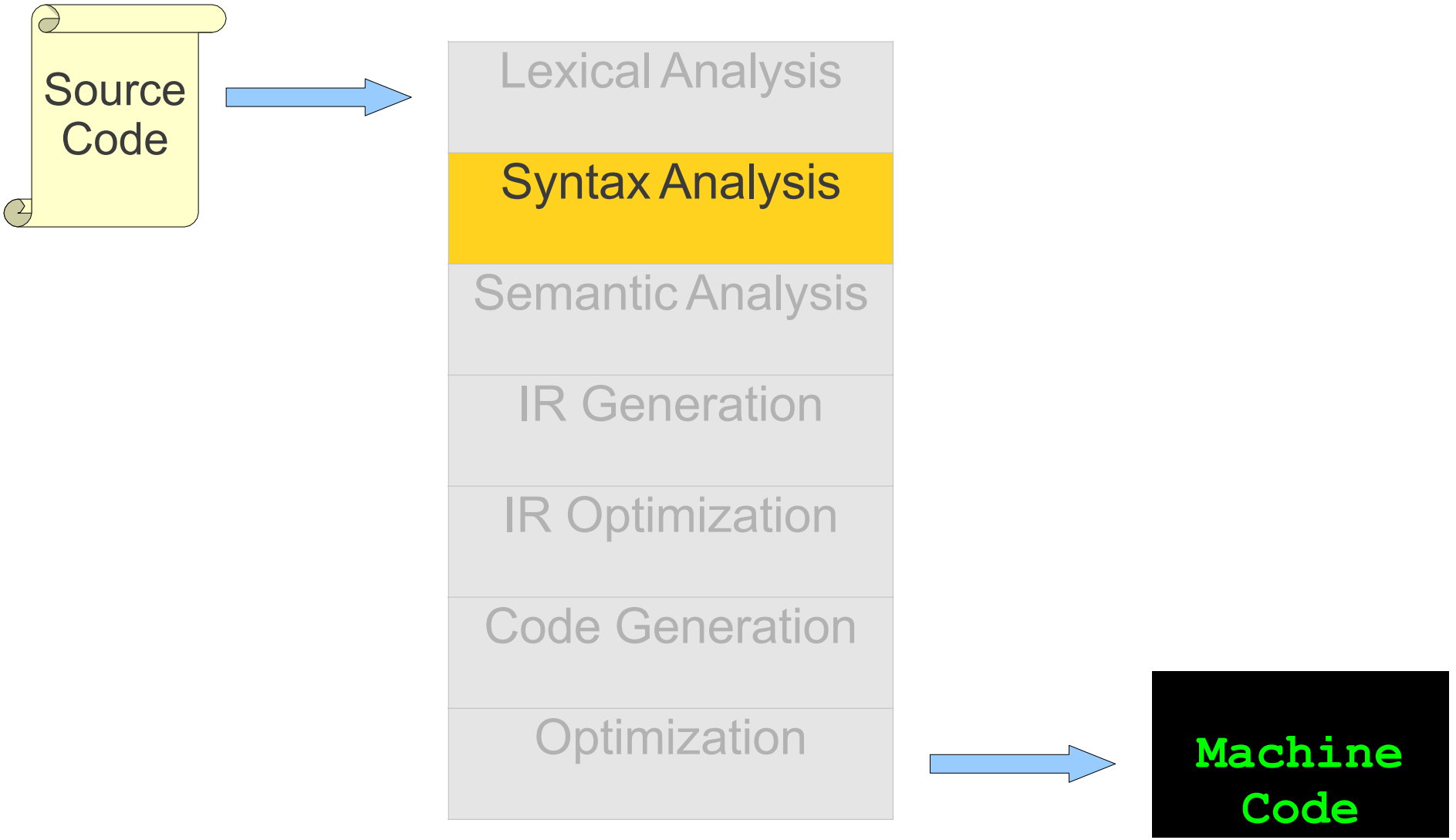
Example: auxiliary functions

```
int installID() {/* function to install the lexeme, whose first
character is pointed to by yytext,
and whose length is yyleng, into the
symbol table and return a pointer thereto */}

int installNum() {/
* similar to installID, but puts numerical constants into a separ
ate table */
}
```

We can use C++ inside **auxiliary function part**

Next Time


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code