Extension of Selected ADFA Construction Algorithms to the Case of Cyclic Automata

Jan Daciuk

Gdańsk University of Technology

e-mail: jandac@eti.pg.gda.pl

Overview

- Incrementality, survey of algorithms for acyclic automata
- Watson's algorithm
- Extension of Watson's algorithm
- Incremental algorithm for sorted data
- Extension of the incremental algorithm for sorted data
- Conclusions

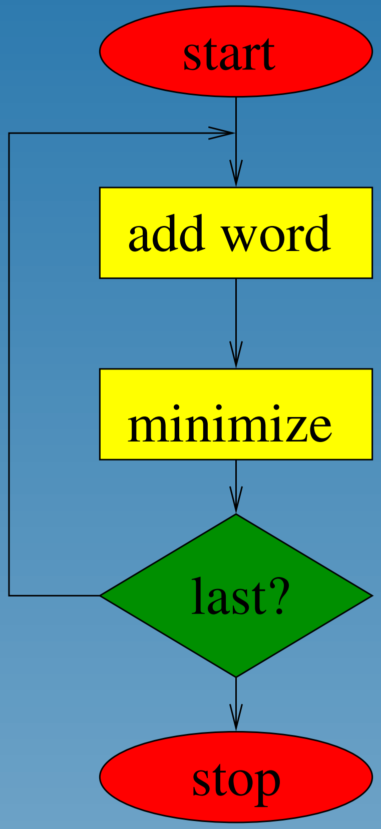
Incrementality and Automata

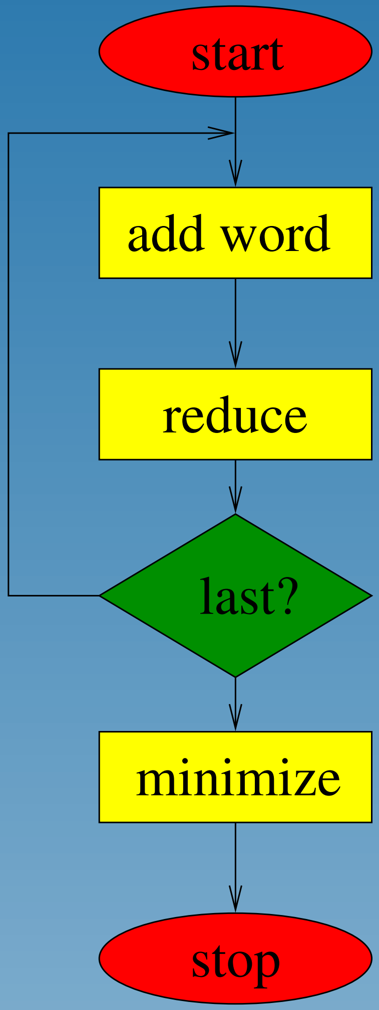
- Finite Automata
  - ◇ ideal for implementation of dictionaries
  - ◇ memory-efficient once constructed
  - ◇ traditional construction needs much memory
- Incremental and semi-incremental construction need less memory
- Moore's law

Traditional Construction

- Acyclic automata
  - ◇ Construct a trie
  - ◇ Minimize it
- Cyclic automata
  - ◇ Construct an NFA
  - ◇ Determinize it
  - ◇ Minimize it

Incremental vs. Semi-Incremental Construction Algorithms





Incremental and semi-incremental algorithms for acyclic automata

- Incremental algorithm for (lexicographically) sorted data (Daciuk, Mihov, Ciura, Deorowicz)
- Incremental algorithm for unsorted data (Aoe, Morimoto, Hase, Sgarbas, Fakotakis, Kokkinakis, Daciuk, Watson, Revuz. . .)
- Semi-incremental algorithm for data lexicographically sorted on reversed strings (Revuz)
- Semi-incremental algorithm for data sorted on decreasing length of strings (Watson)

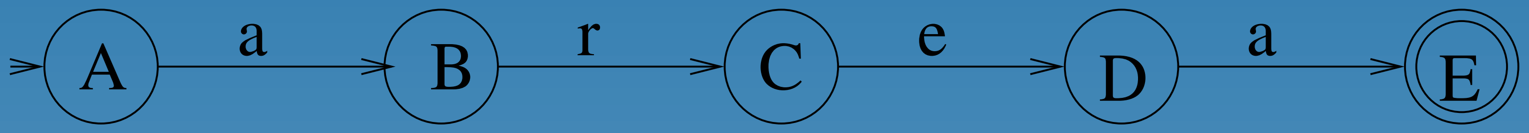
Extension by Carrasco and Forcada

- There is a pre-existing cyclic automaton
- All states in the pre-existing automaton are already in the **register**
- When adding new words, all states in the **common prefix** path are **cloned**, including the start state
- Old start state and some of its descendants are removed if they became unreachable

Watson's Semi-Incremental Algorithm

- Find the common prefix
- Create a chain of states and transitions for the suffix – make the last state final
- Find all states reachable from the final state up to and excluding any final states; put them onto stack as they are being found
- For all states on stack, replace them with equivalent states or put them into the register

Watson's Algorithm – Example

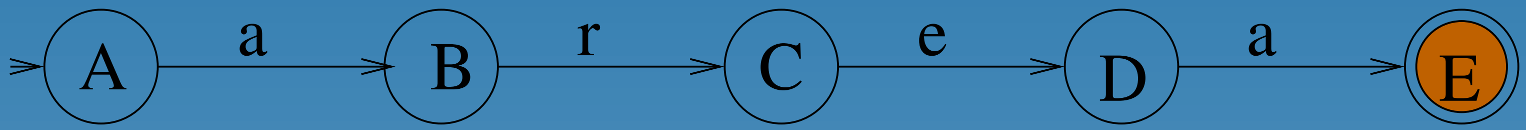


$$\begin{tikzcd} \arrow[r] & A \arrow[r, "a"] & B \arrow[r, "r"] & C \arrow[r, "e"] & D \arrow[r, "a"] & E \end{tikzcd}$$

Register = { } Stack = {E}

area

Watson's Algorithm – Example

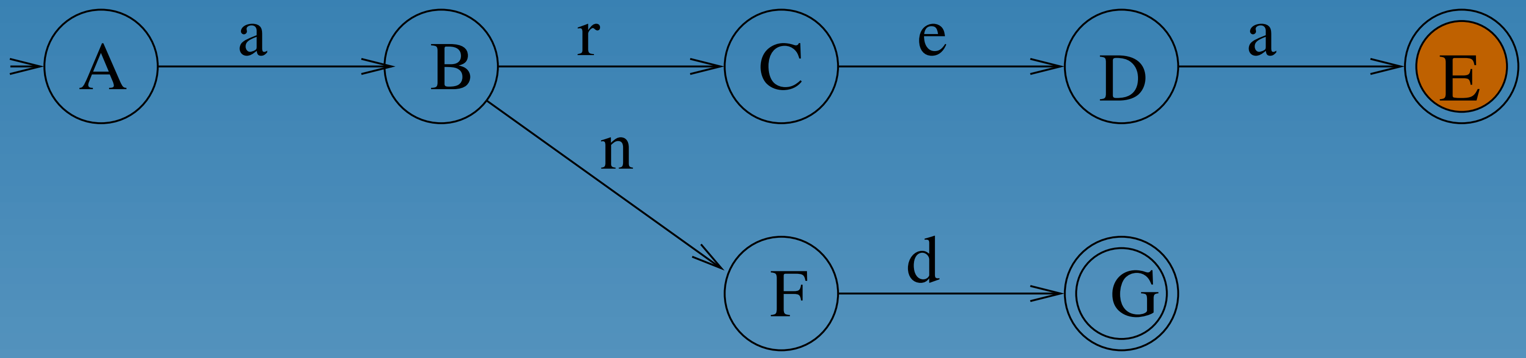

$$\begin{tikzcd} \arrow[r] & A \arrow[r, "a"] & B \arrow[r, "r"] & C \arrow[r, "e"] & D \arrow[r, "a"] & E \end{tikzcd}$$

Register = {E}

Stack = { }

area

Watson's Algorithm – Example

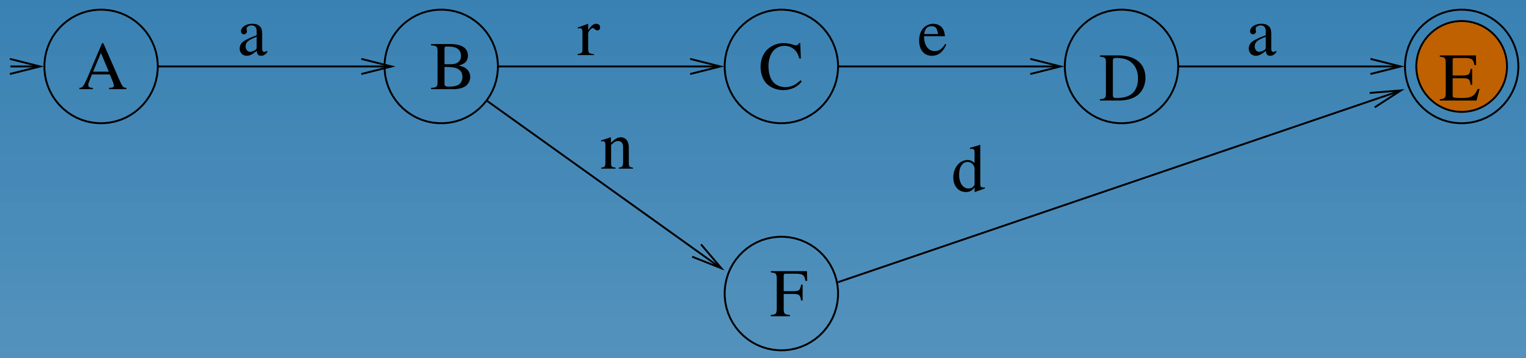


Register = {E}

Stack = {G}

and

Watson's Algorithm – Example

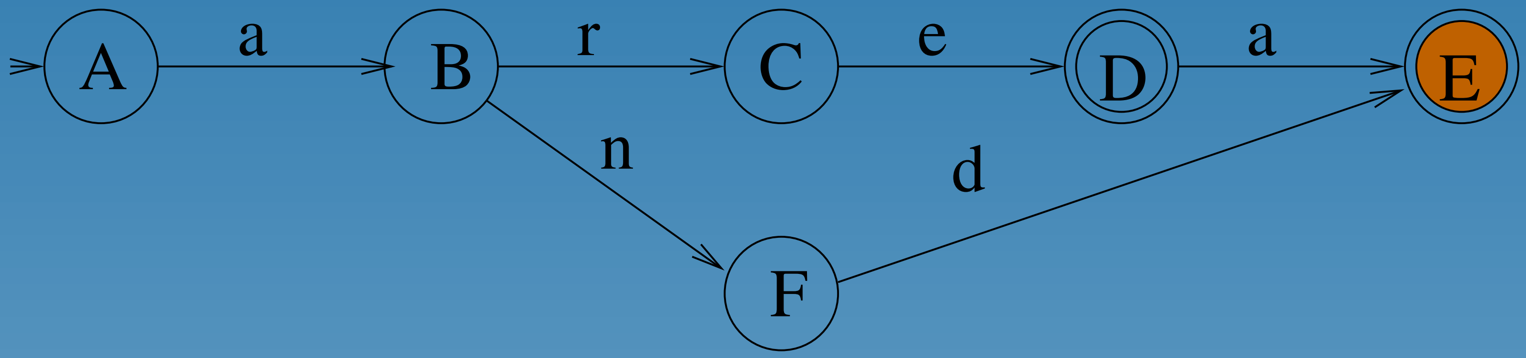


Register = {E}

Stack = { }

and

Watson's Algorithm – Example

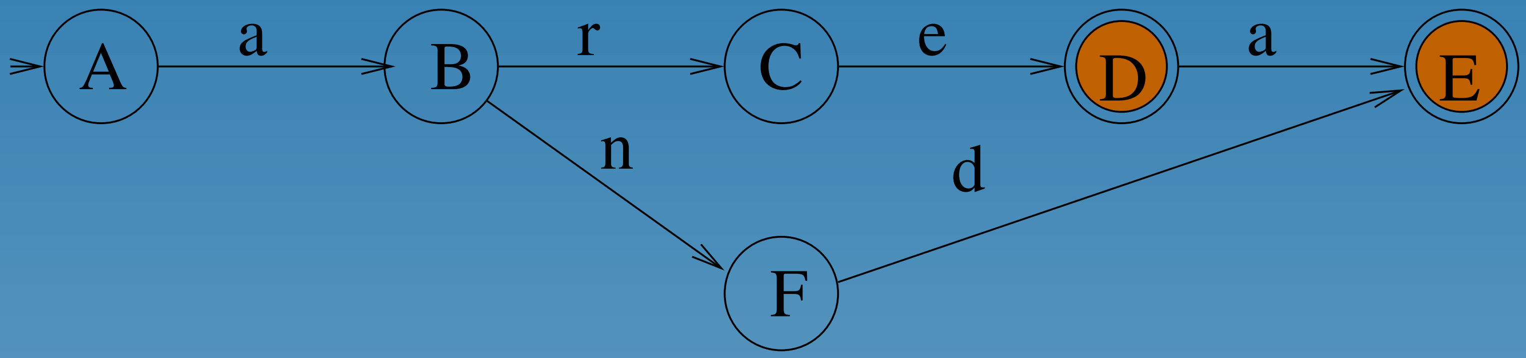


Register = {E}

Stack = {D}

are

Watson's Algorithm – Example

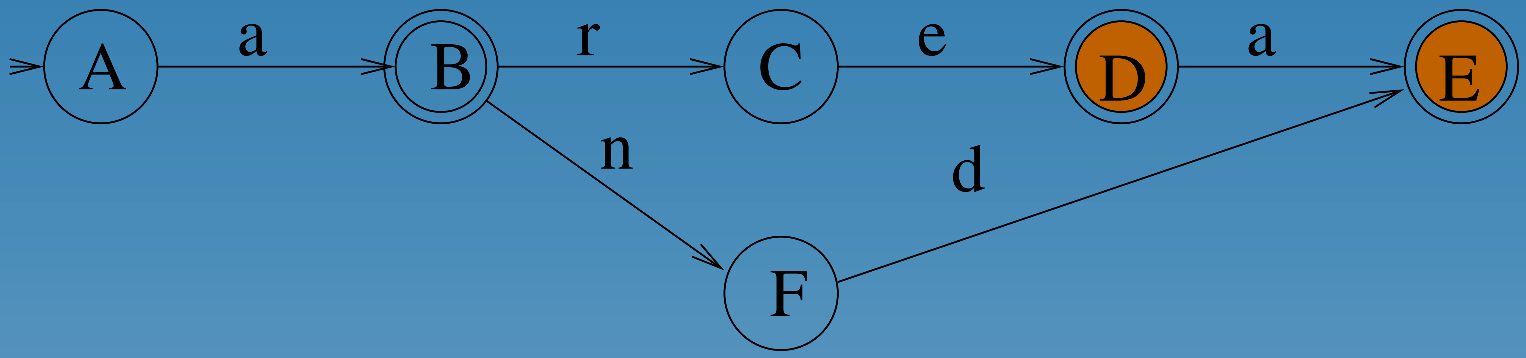


Register = {E,D}

Stack = { }

are

Watson's Algorithm – Example

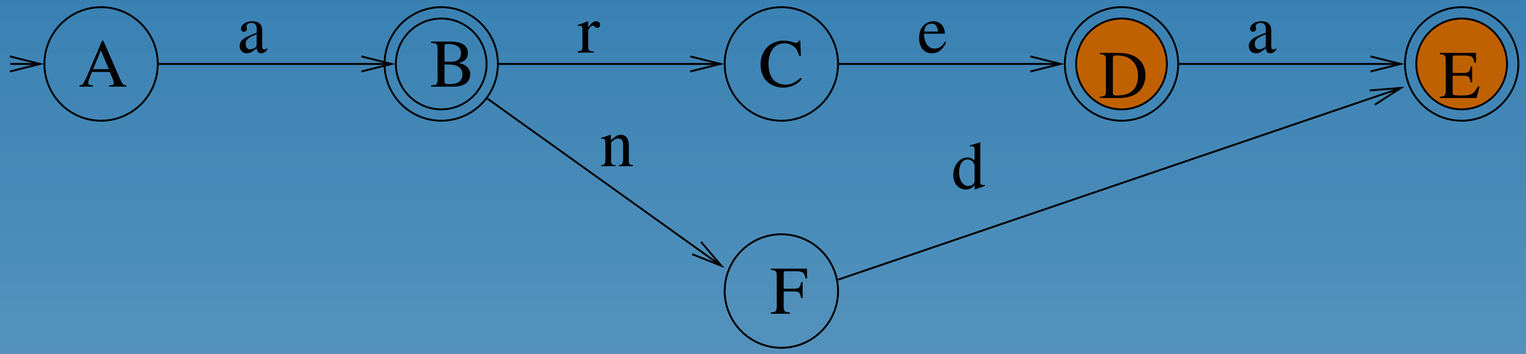


Register = {E,D}

Stack = {B}

a

Watson's Algorithm – Example

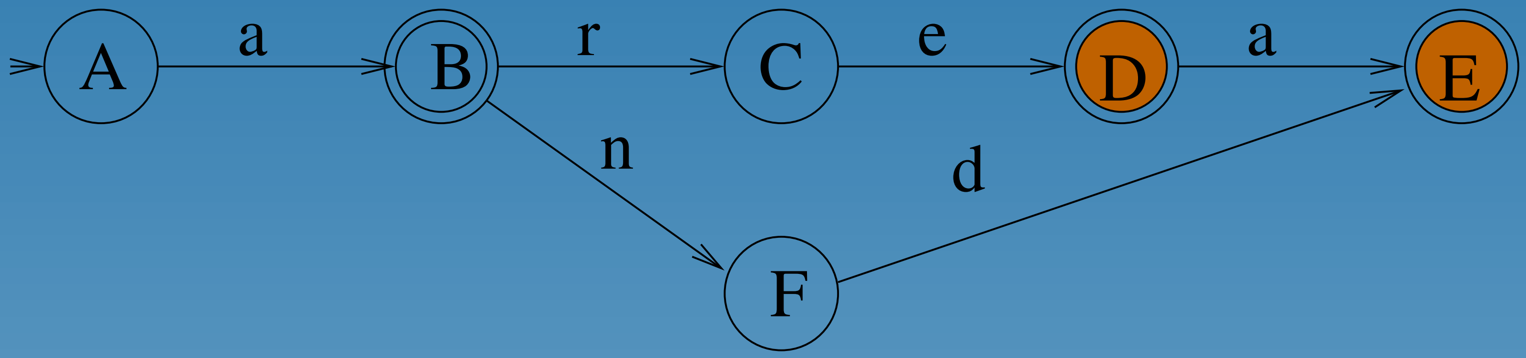


Register = {E,D}

Stack = {B,C}

a

Watson's Algorithm – Example

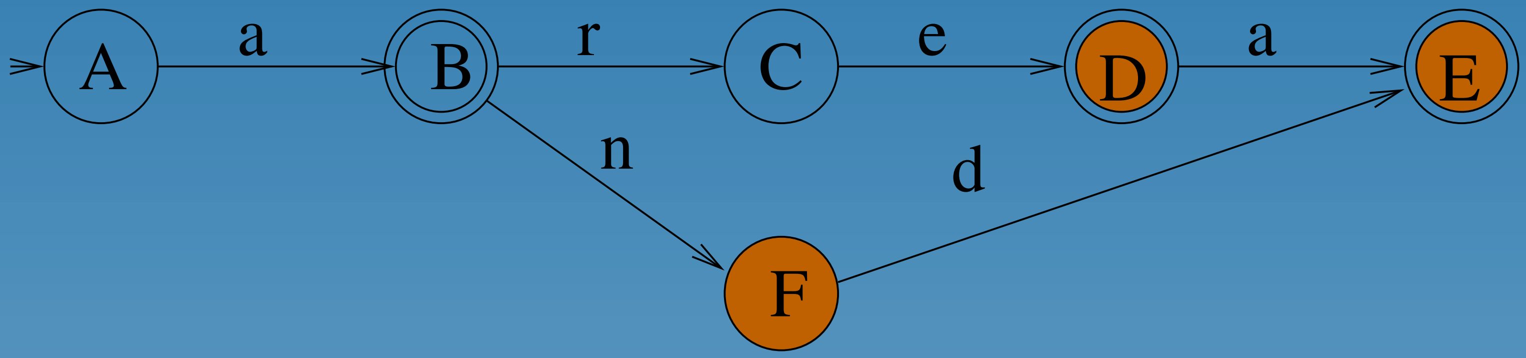


Register = {E,D}

Stack = {B,C,F}

a

Watson's Algorithm – Example

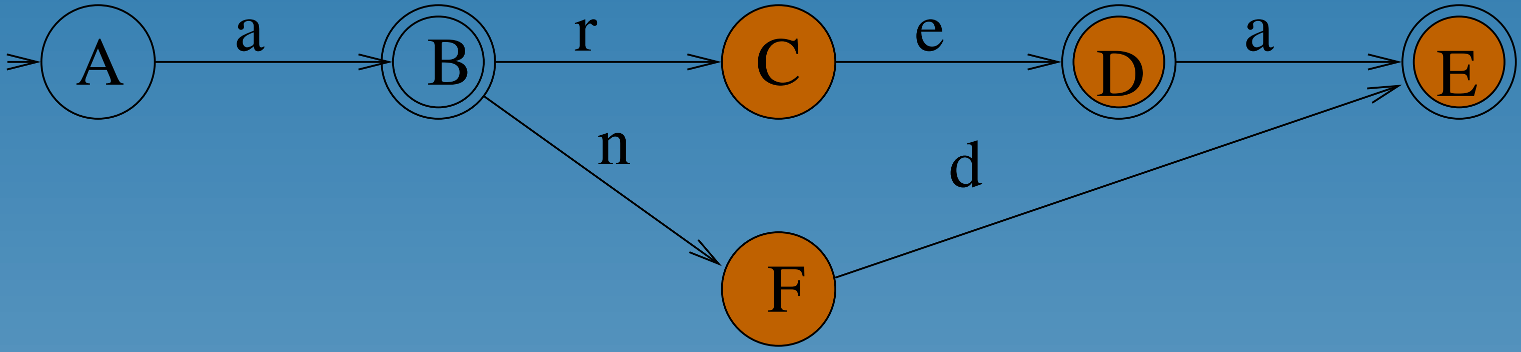


Register = {E,D,F}

Stack = {B,C}

a

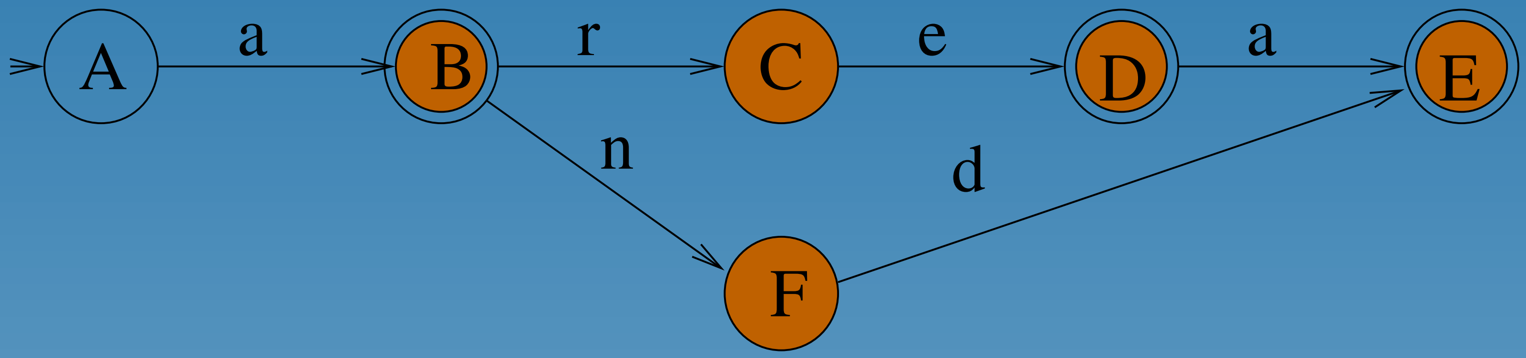
Watson's Algorithm – Example



Register = {E,D,F,C}   Stack = {B}

a

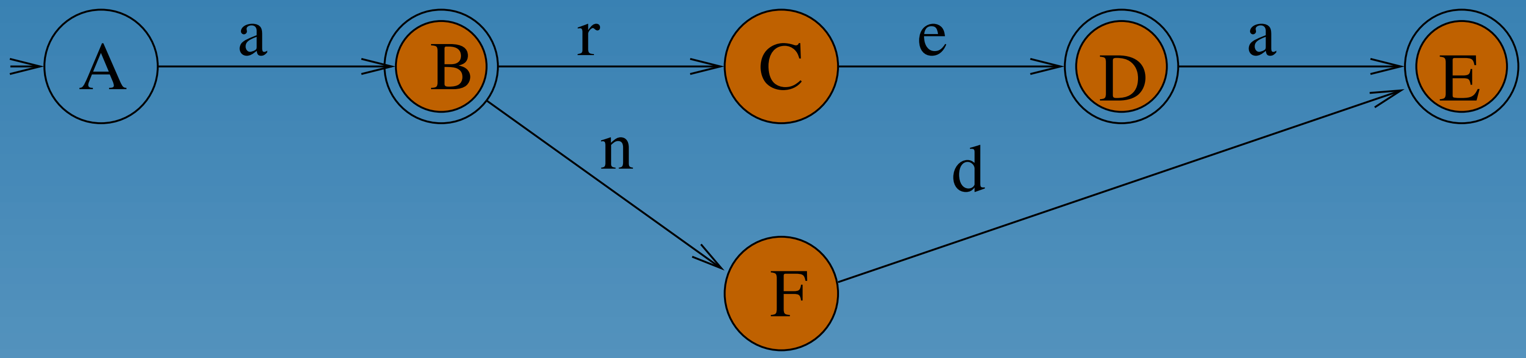
Watson's Algorithm – Example



Register = {E,D,F,C,B}   Stack = { }

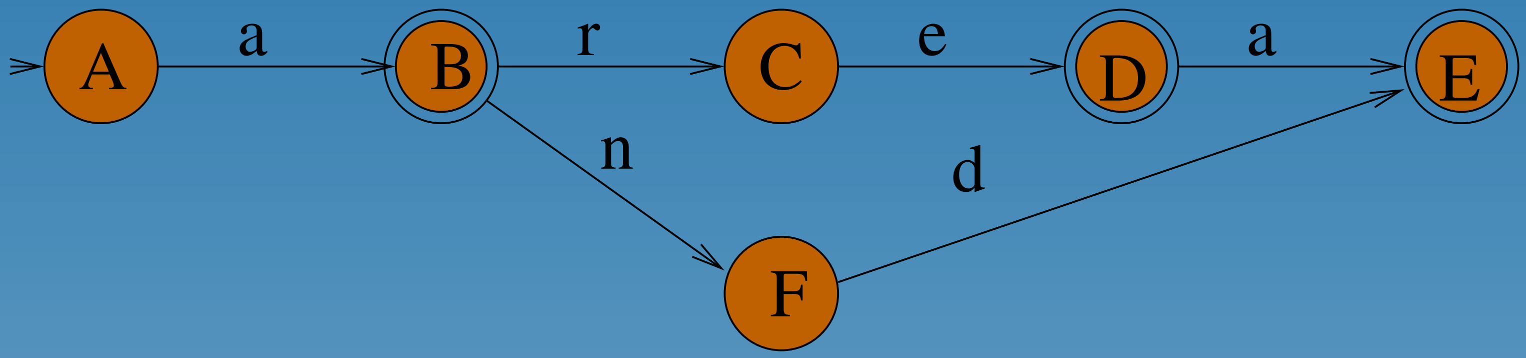
a

Watson's Algorithm – Example



Register = {E,D,F,C,B}   Stack = {A}

Watson's Algorithm – Example



Register = {E,D,F,C,B,A}   Stack = { }

Extension of Watson's Algorithm to Cyclic Automata

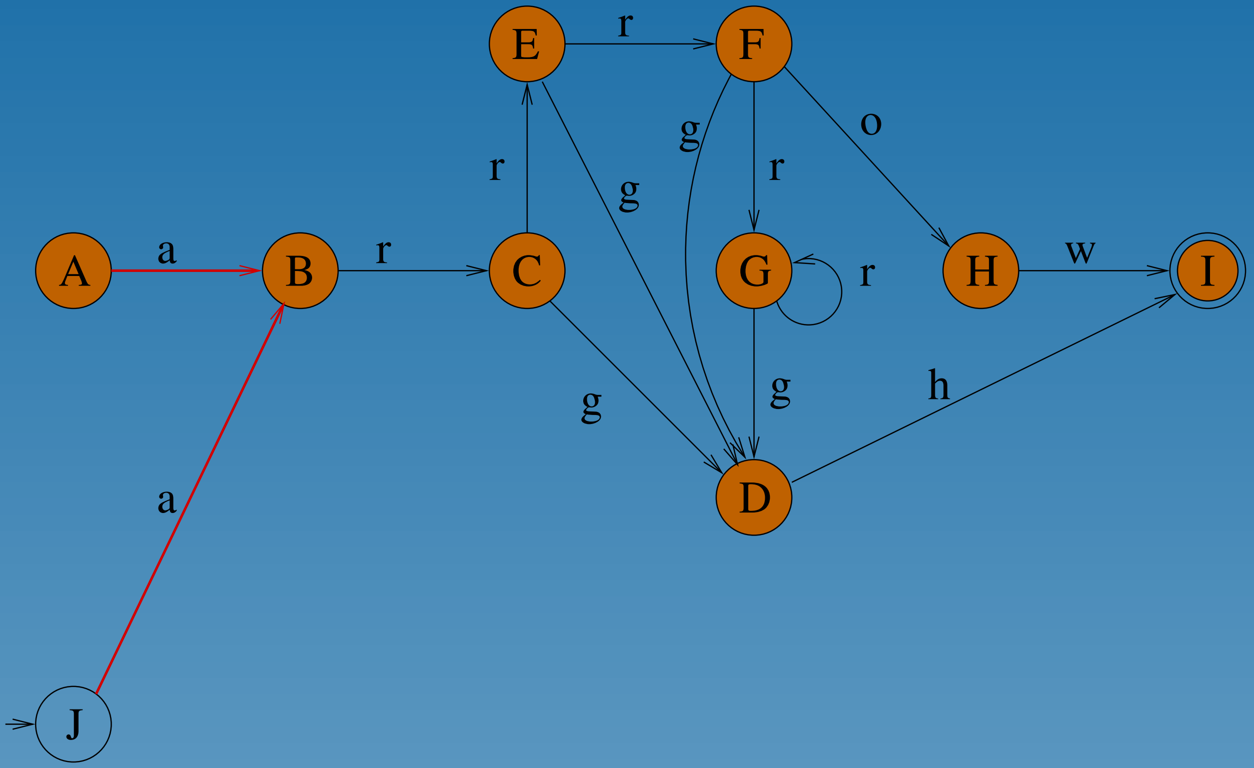
```
1:   function add_word(w)
2:     $q = q_0; i \leftarrow 0;$
3:     while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
4:       $q \leftarrow \delta(q, w_i); i \leftarrow i + 1;$
5:     end while
6:     while $i < |w|$ do
7:       $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
8:     end while
9:     $F \leftarrow F \cup \{q\};$
10:    return $q$;
11:  end function
```

```
1:   function add_word(w)
2:     $q = q_0; i \leftarrow 0;$
3':    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
           and $fanin(\delta(q, w_i)) < 2$ do
4':      $q \leftarrow \delta(q, w_i);\ i \leftarrow i + 1$
5':    end while
3'':   while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
4'':     $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
4''':    $q \leftarrow \delta(q, w_i);\ i \leftarrow i + 1$
5'':   end while
6:     while $i < |w|$ do
7:       $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
8:     end while
9:     $F \leftarrow F \cup \{q\};$
10:    return $q$;
11:  end function
```

- Clone the initial state and clone any confluence (reentrant) states in the common prefix path
- When putting states onto stack, stop before final and **confluence** states

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                    and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```



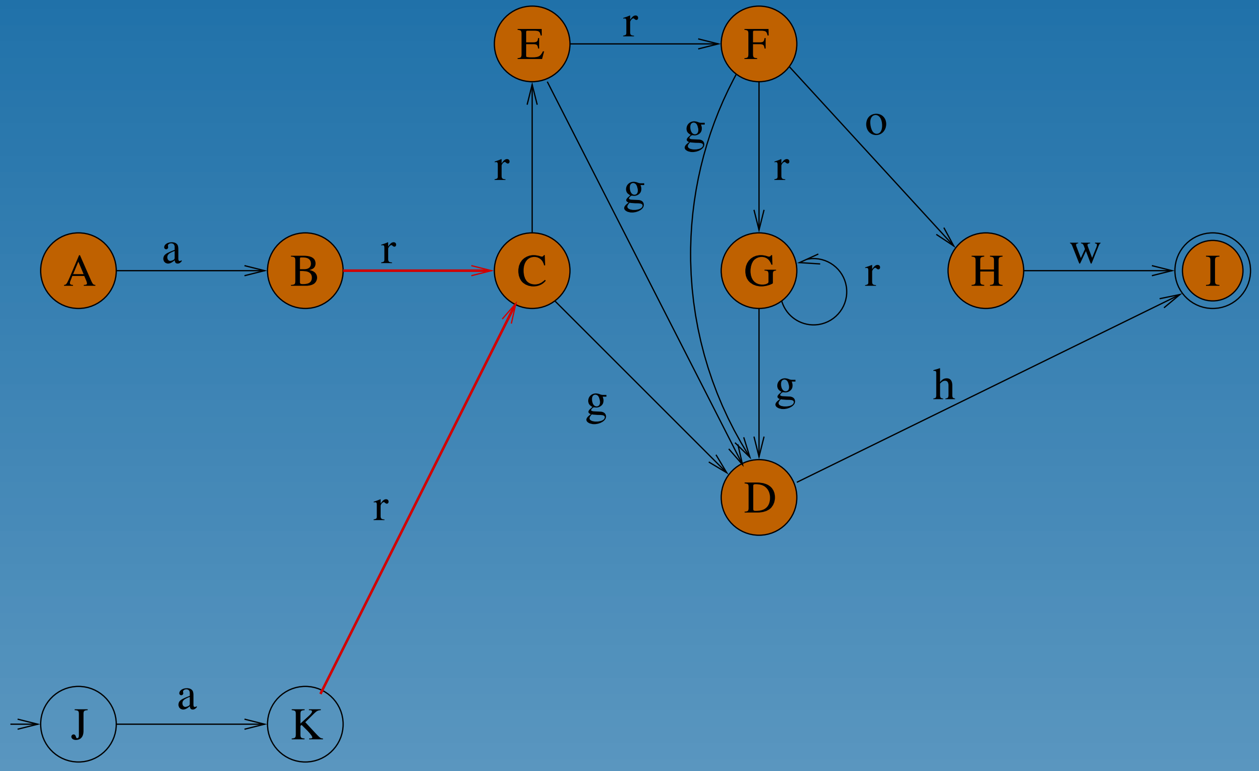
Register = {A,B,C,D,E,F,G,H,I}

area

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0$; $i \leftarrow 0$;
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
  end while
  $F \leftarrow F \cup \{q\}$;
  return $q$;
end function
```



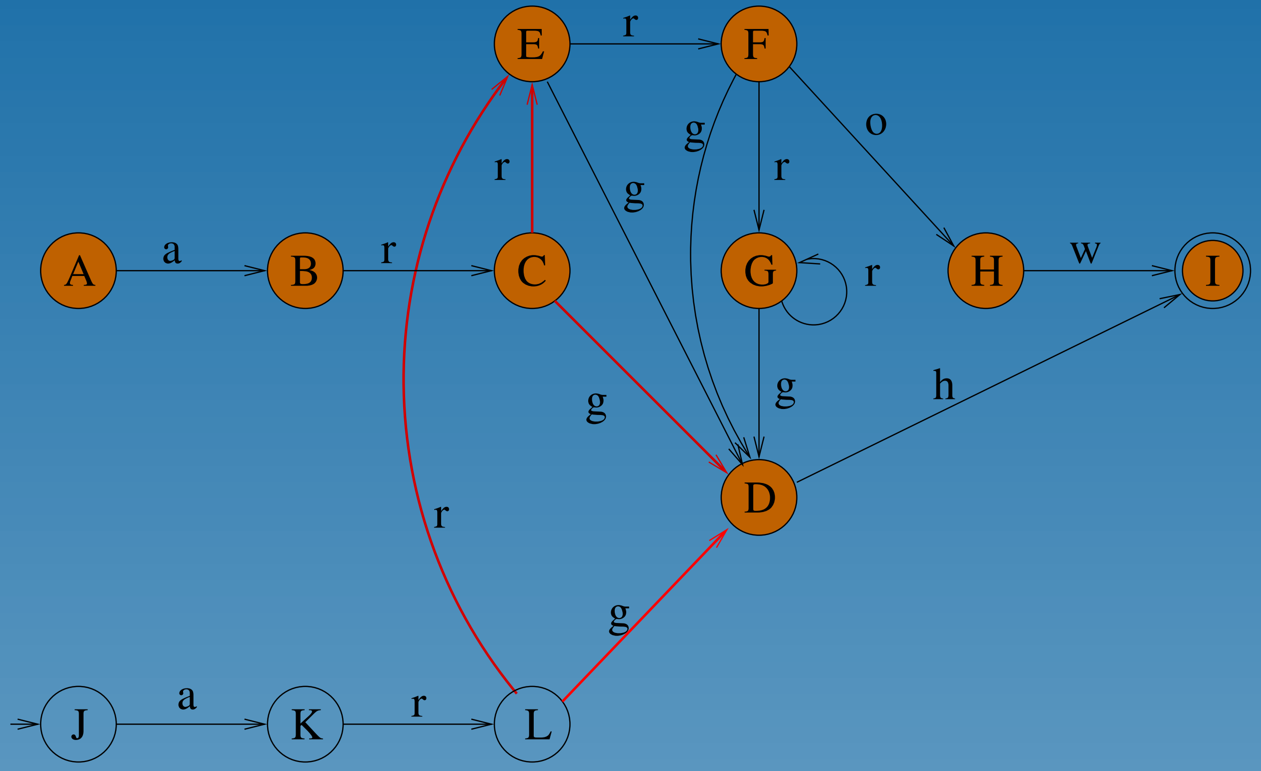
Register = {A,B,C,D,E,F,G,H,I}

area

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0$; $i \leftarrow 0$;
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
  end while
  $F \leftarrow F \cup \{q\}$;
  return $q$;
end function
```



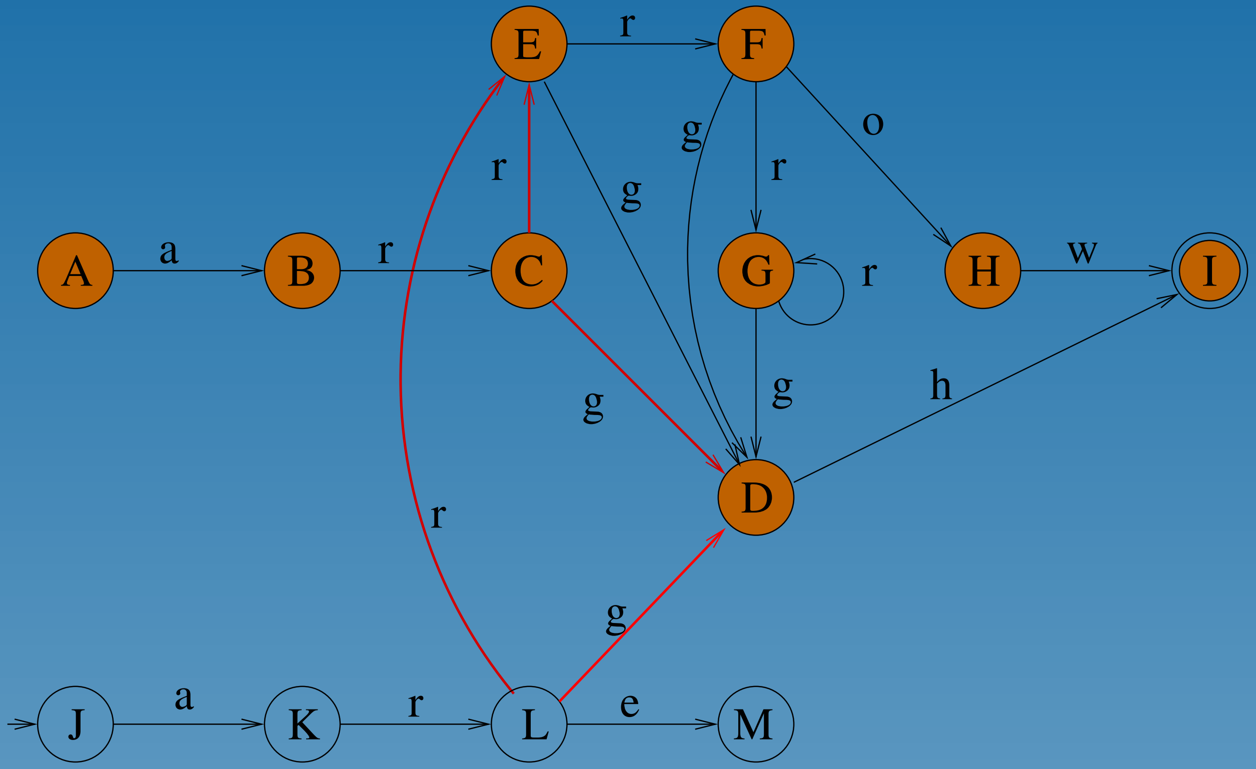
Register = {A,B,C,D,E,F,G,H,I}

area

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```



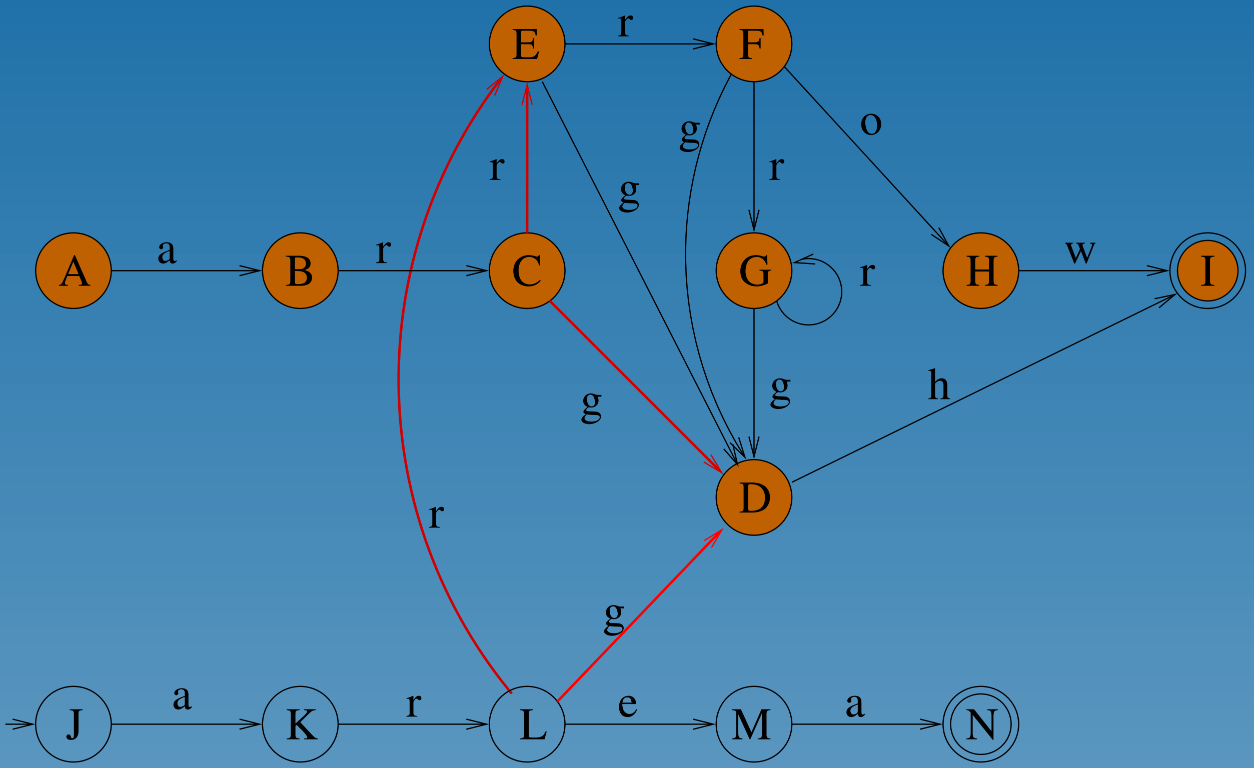
Register = {A,B,C,D,E,F,G,H,I}

area

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
   $q = q_0$; $i \leftarrow 0$;
   while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                  and $fanin(\delta(q, w_i)) < 2$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
   end while
   while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
      $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
   end while
   while $i < |w|$ do
      $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
   end while
   $F \leftarrow F \cup \{q\}$;
   return $q$;
end function
```



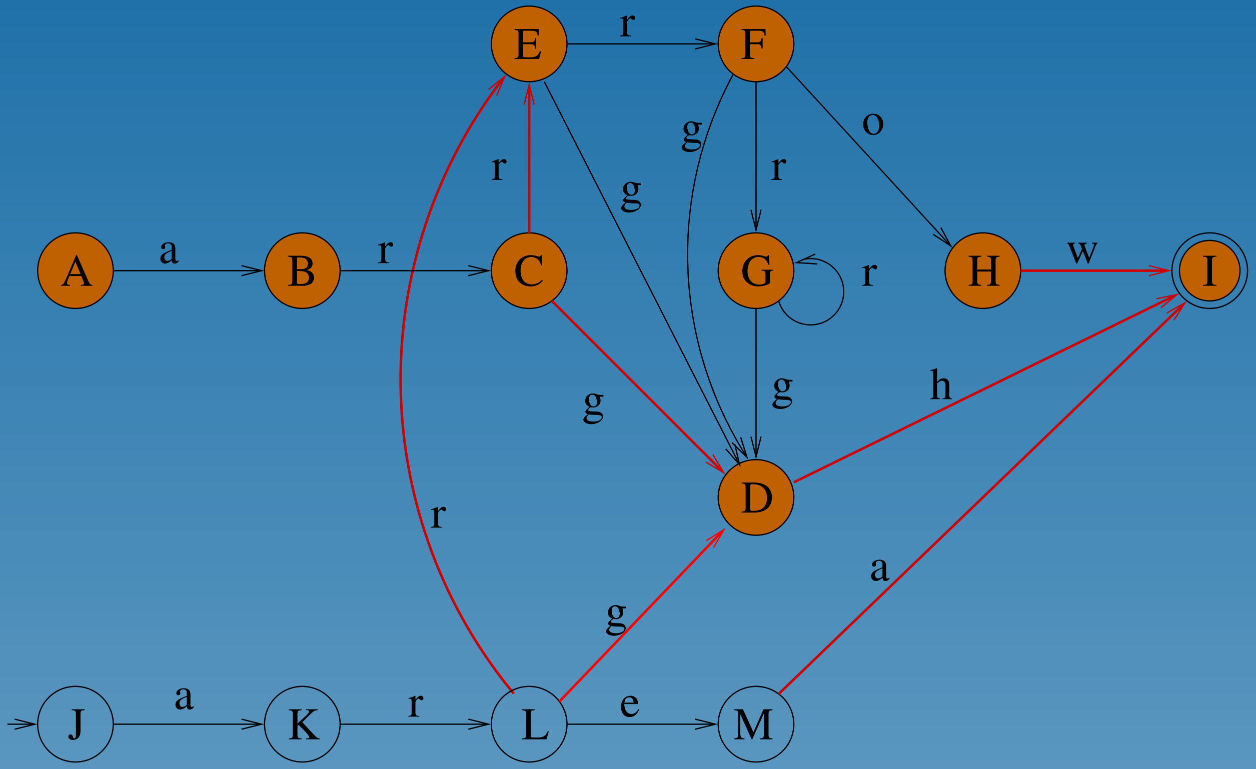
Register = {A,B,C,D,E,F,G,H,I}

area

Stack = {N}

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                    and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```



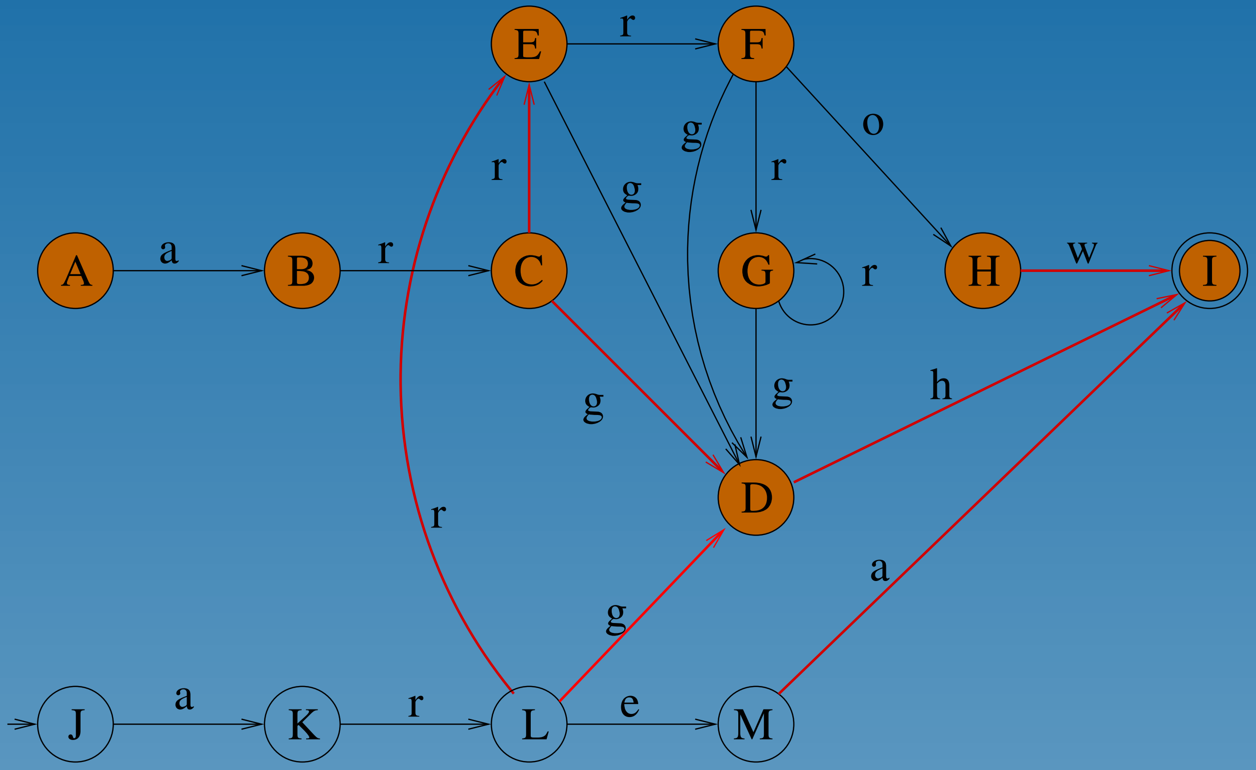
Register = {A,B,C,D,E,F,G,H,I}

area

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                  and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```

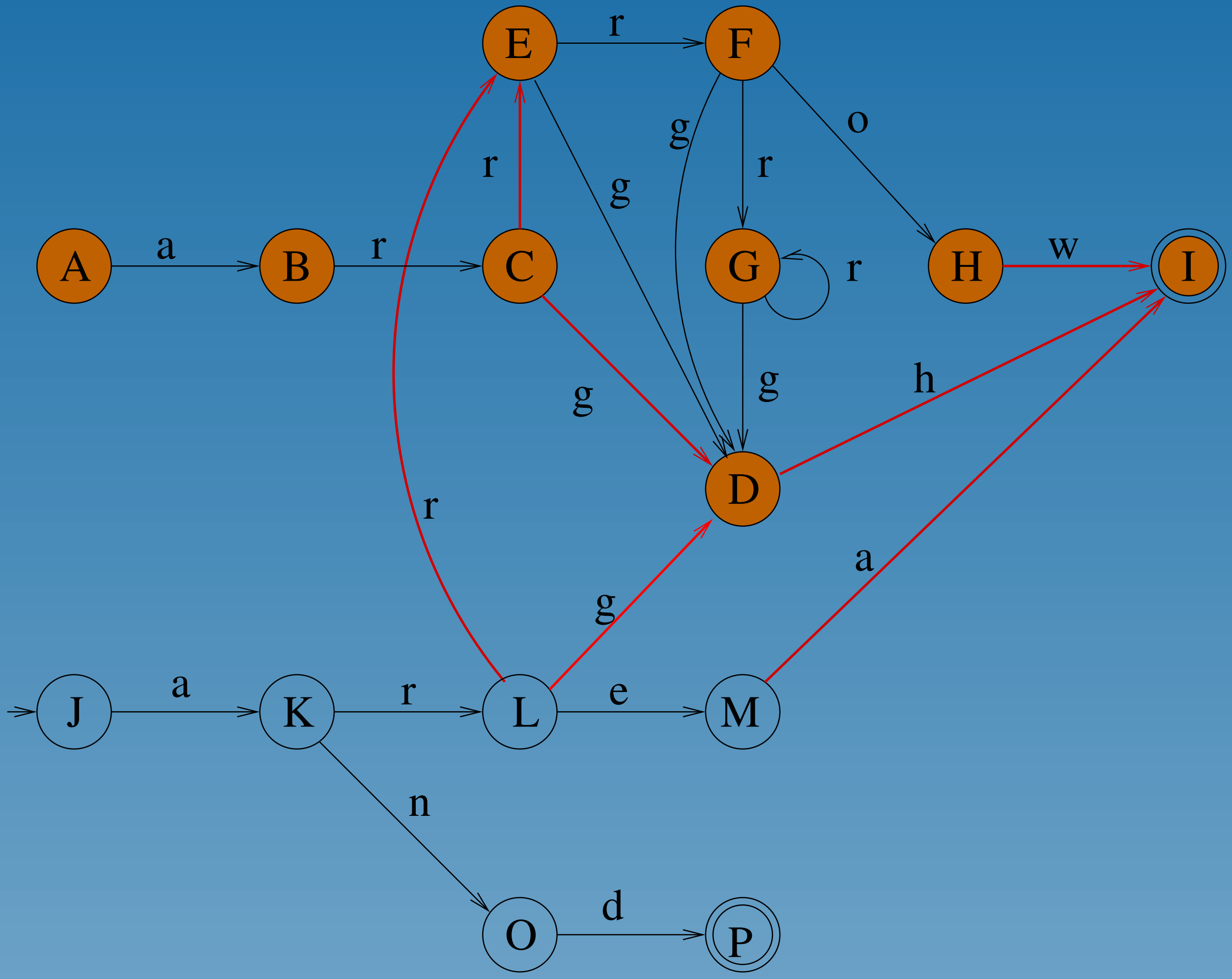


Register = {A,B,C,D,E,F,G,H,I} and

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```

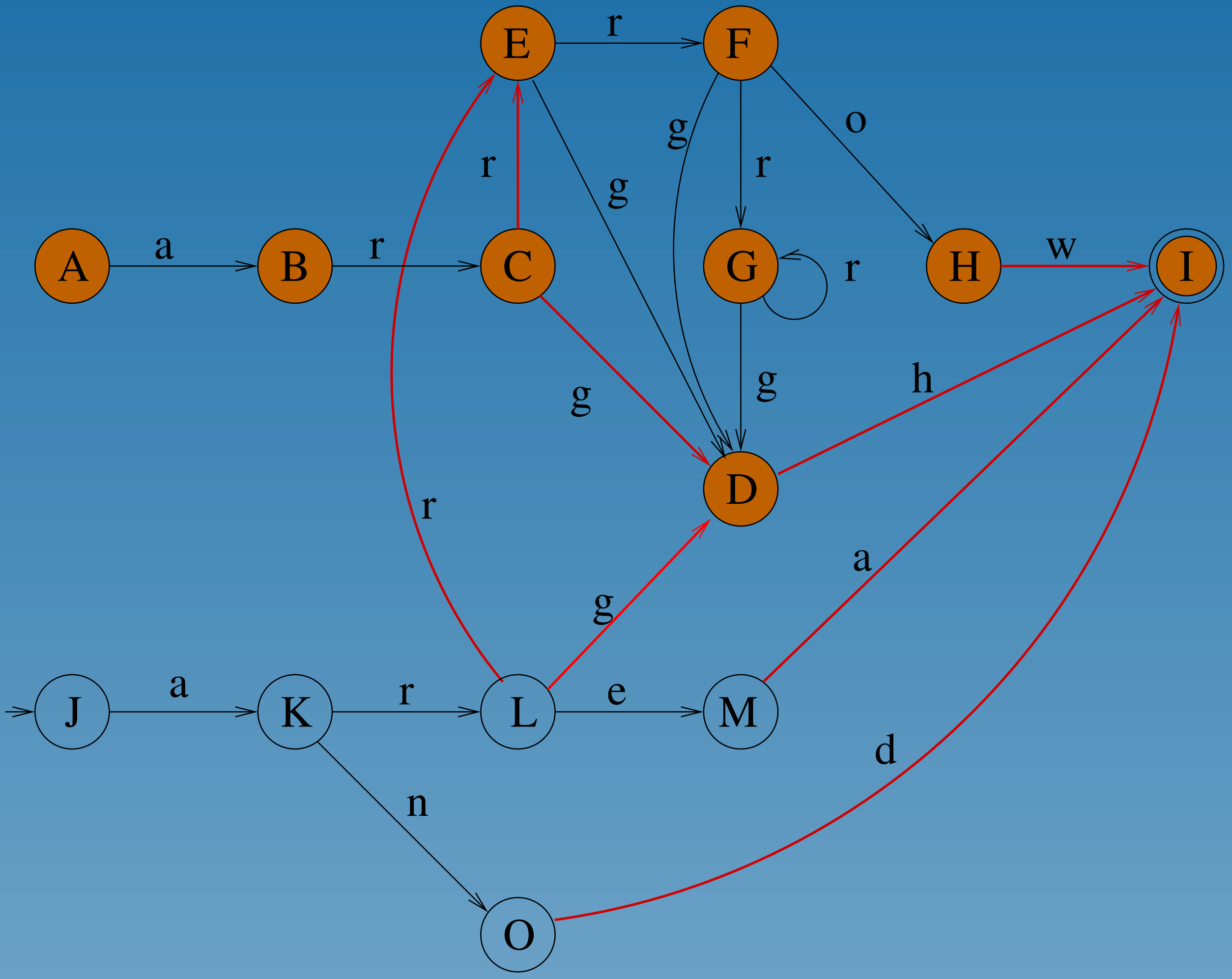


Register = {A,B,C,D,E,F,G,H,I} and

Stack = {P}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0$; $i \leftarrow 0$;
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
  end while
  $F \leftarrow F \cup \{q\}$;
  return $q$;
end function
```

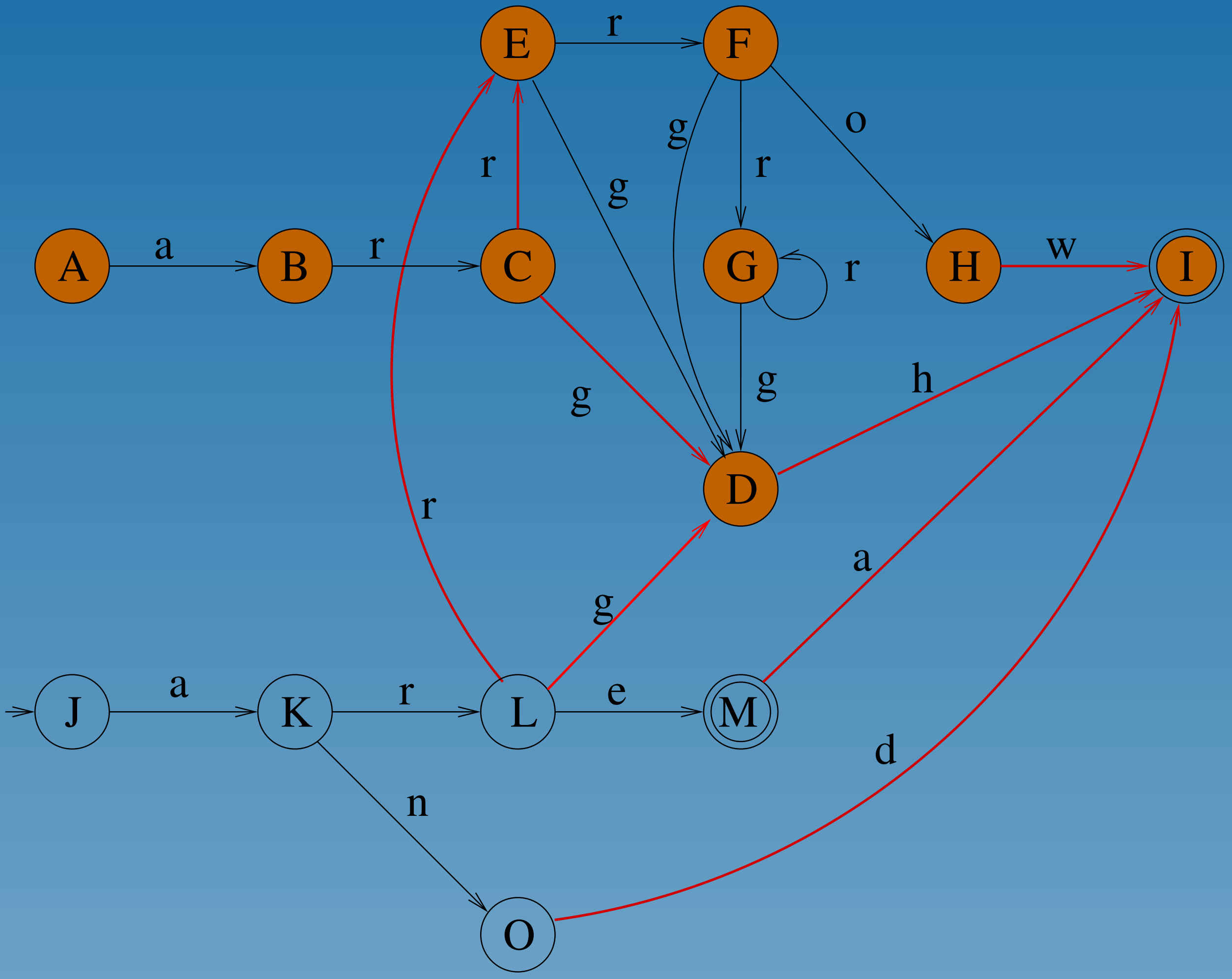


Register = {A,B,C,D,E,F,G,H,I} and

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                  and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```

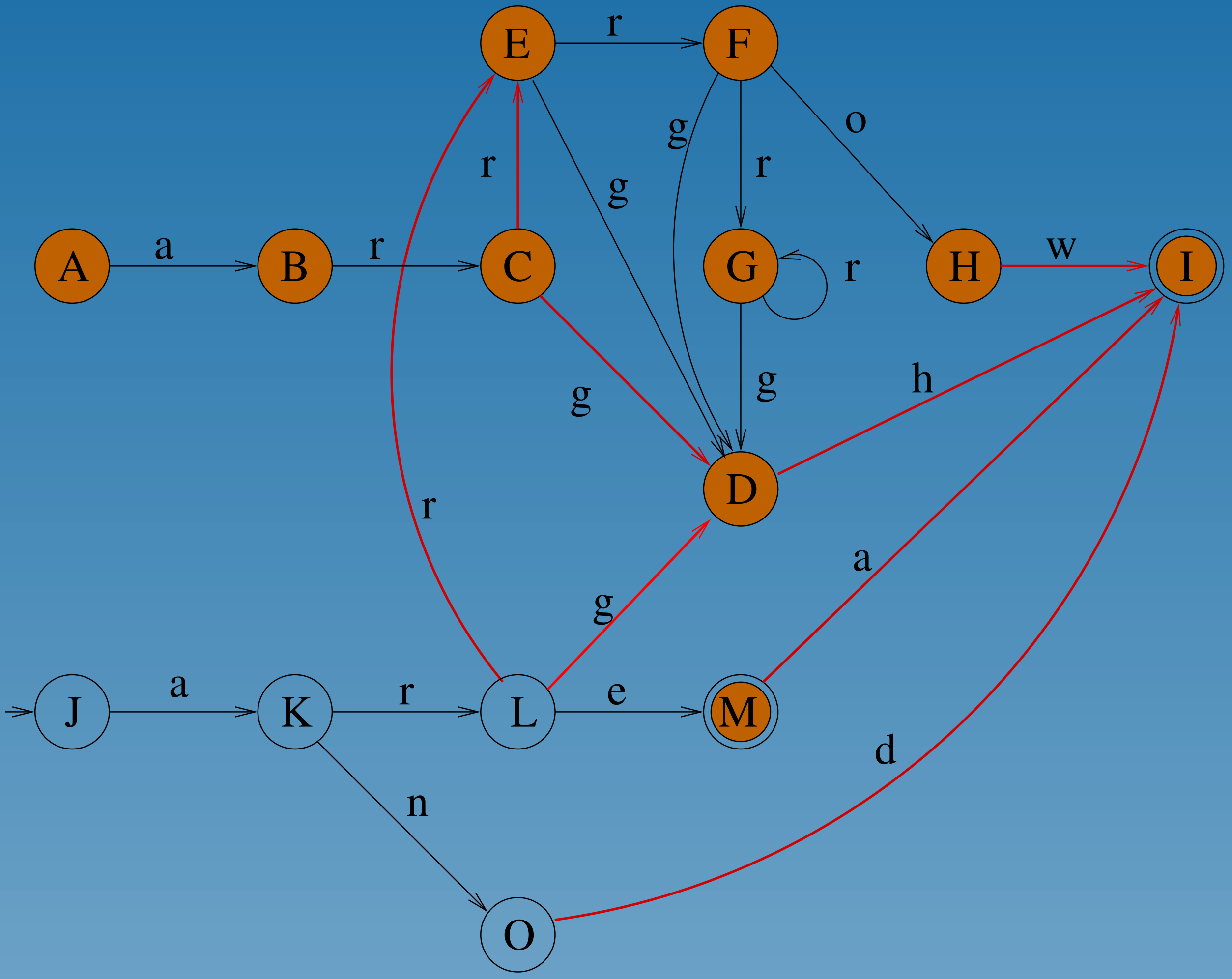


Register = {A,B,C,D,E,F,G,H,I} are

Stack = {M}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0$; $i \leftarrow 0$;
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
  end while
  $F \leftarrow F \cup \{q\}$;
  return $q$;
end function
```

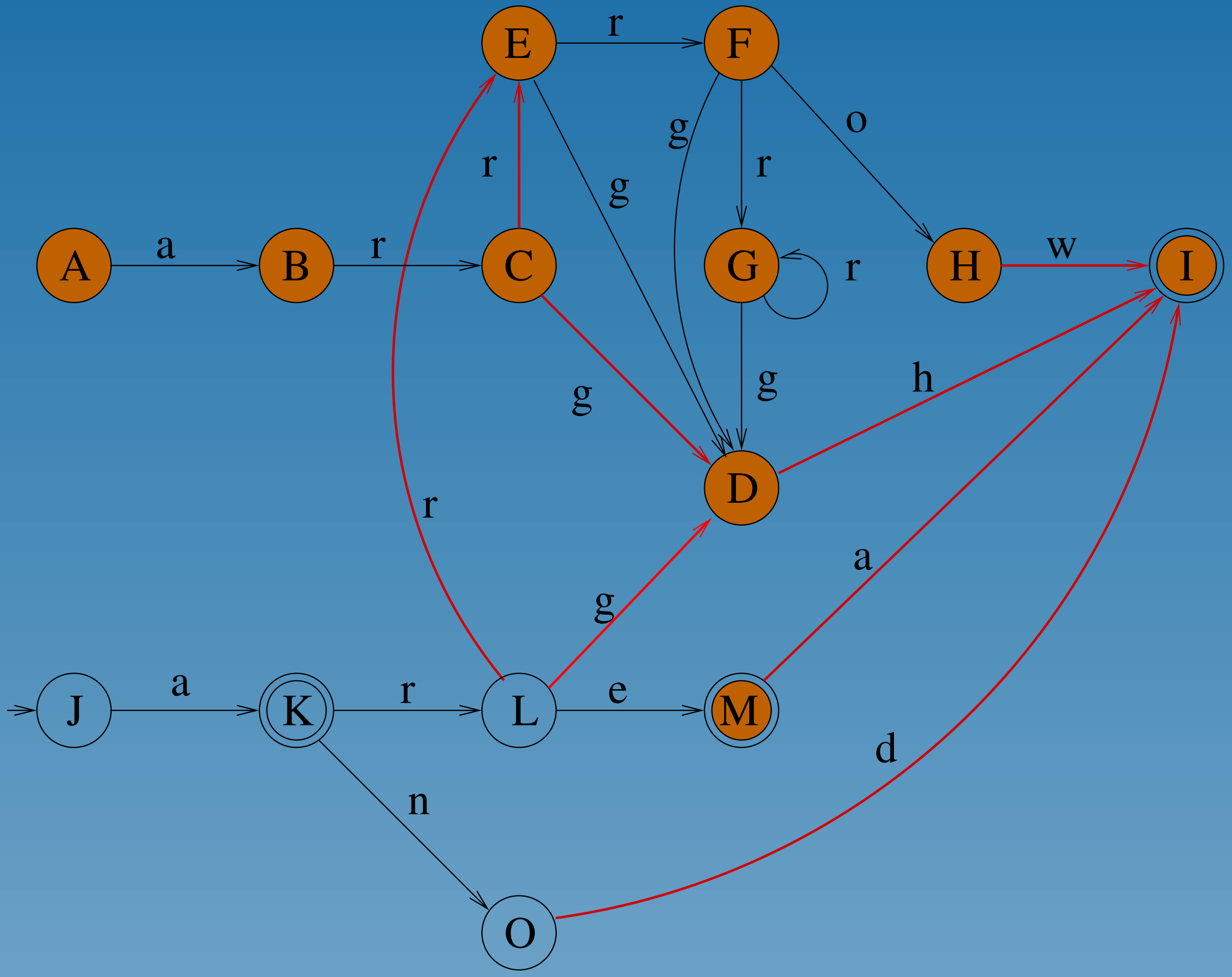


Register = {A,B,C,D,E,F,G,H,I,M} are

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                 and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```

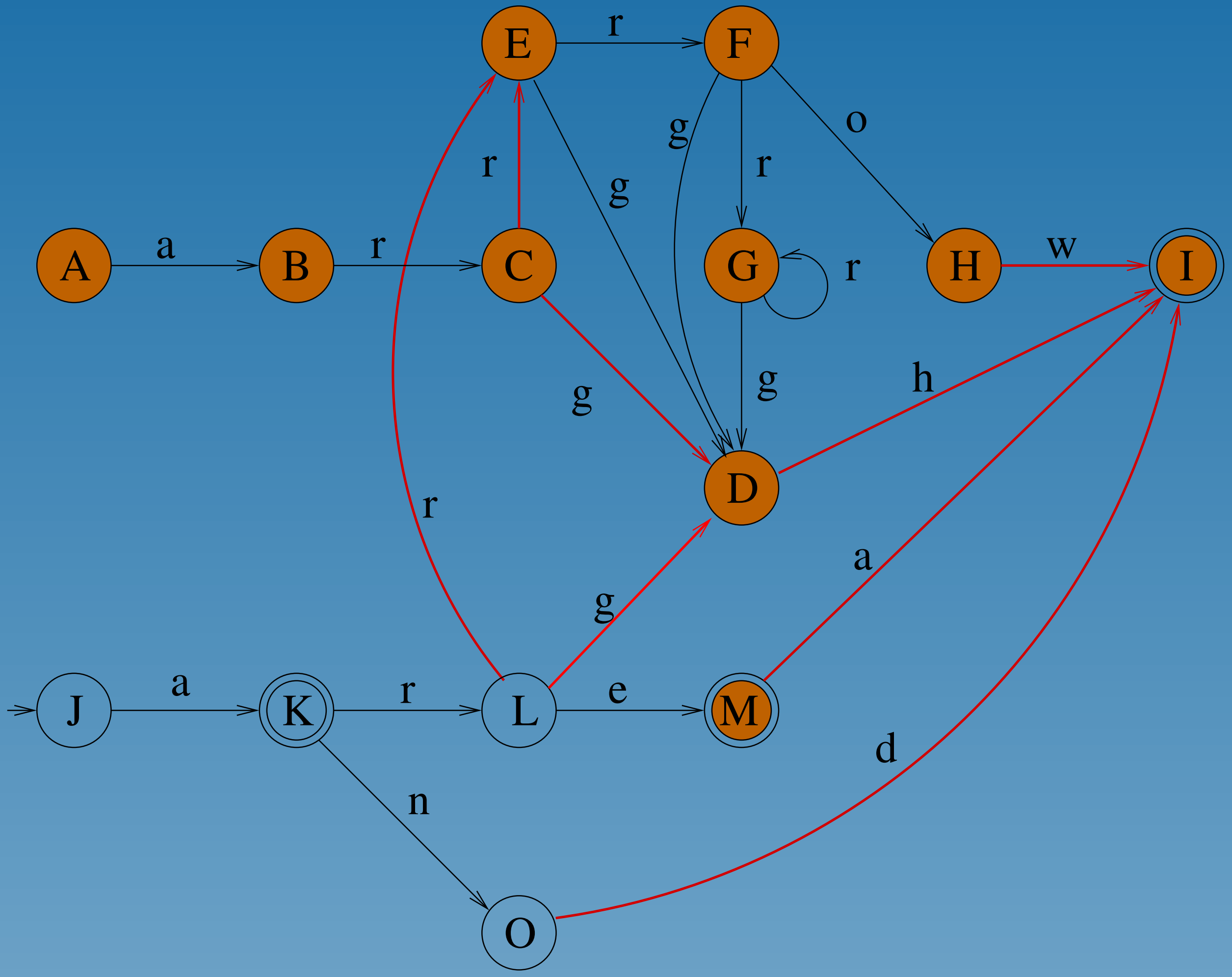


Register = {A,B,C,D,E,F,G,H,I,M}   a

Stack = {K}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0$; $i \leftarrow 0$;
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
  end while
  $F \leftarrow F \cup \{q\}$;
  return $q$;
end function
```



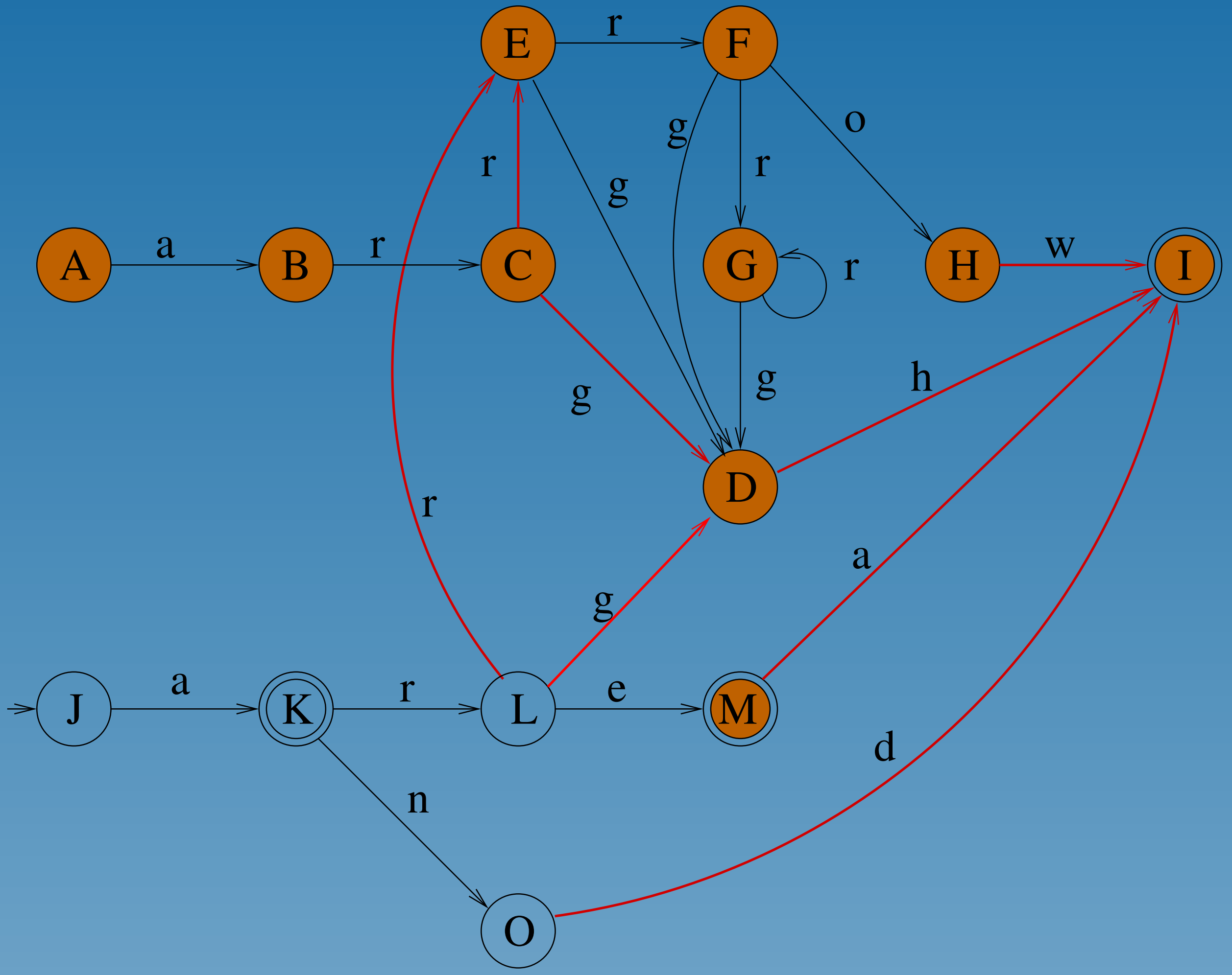
Register = {A,B,C,D,E,F,G,H,I,M}

a

Stack = {K,L}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0; i \leftarrow 0;$
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
  end while
  $F \leftarrow F \cup \{q\};$
  return $q$;
end function
```

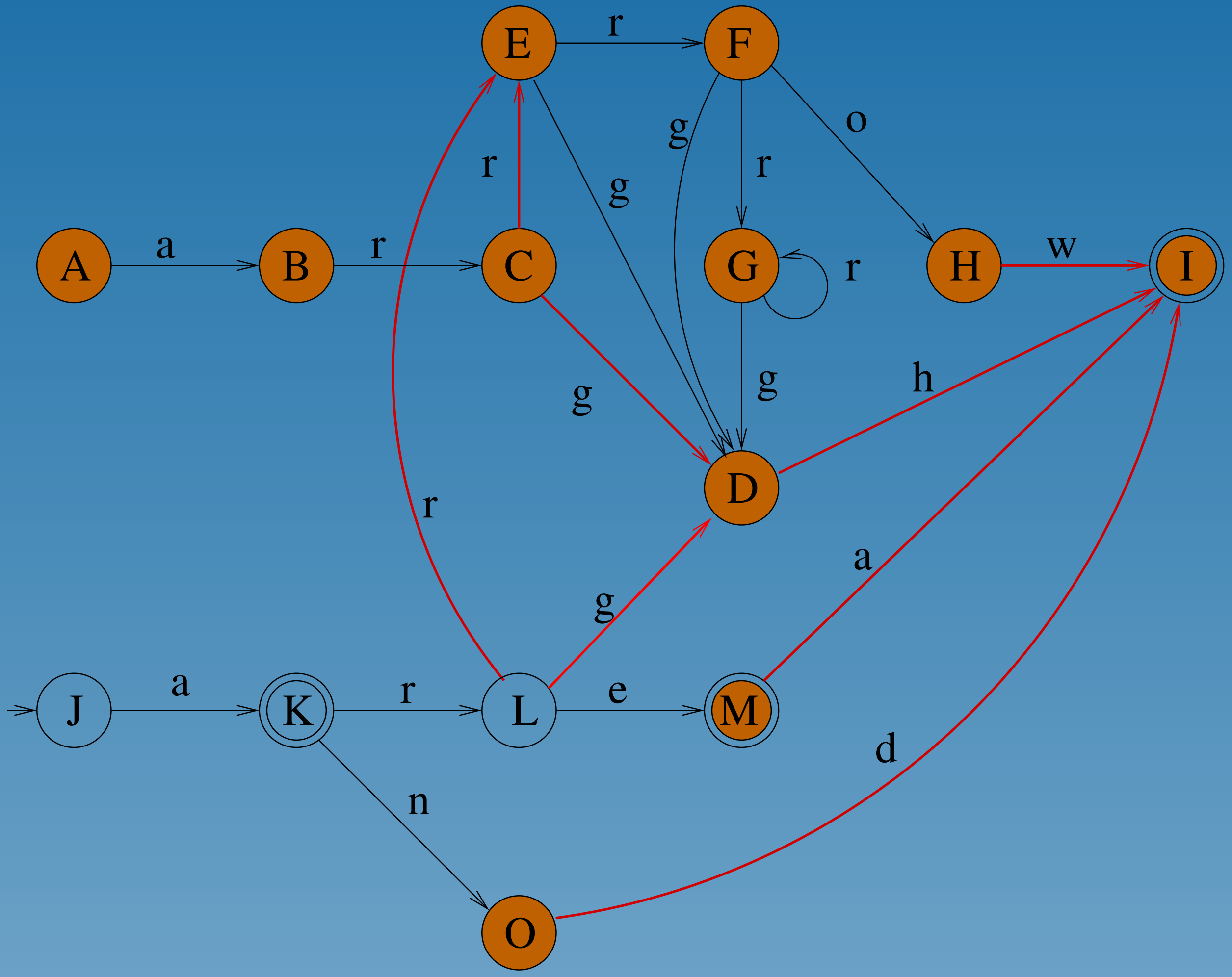


Register = {A,B,C,D,E,F,G,H,I,M}   a

Stack = {K,L,O}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0; i \leftarrow 0;$
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
  end while
  $F \leftarrow F \cup \{q\};$
  return $q$;
end function
```



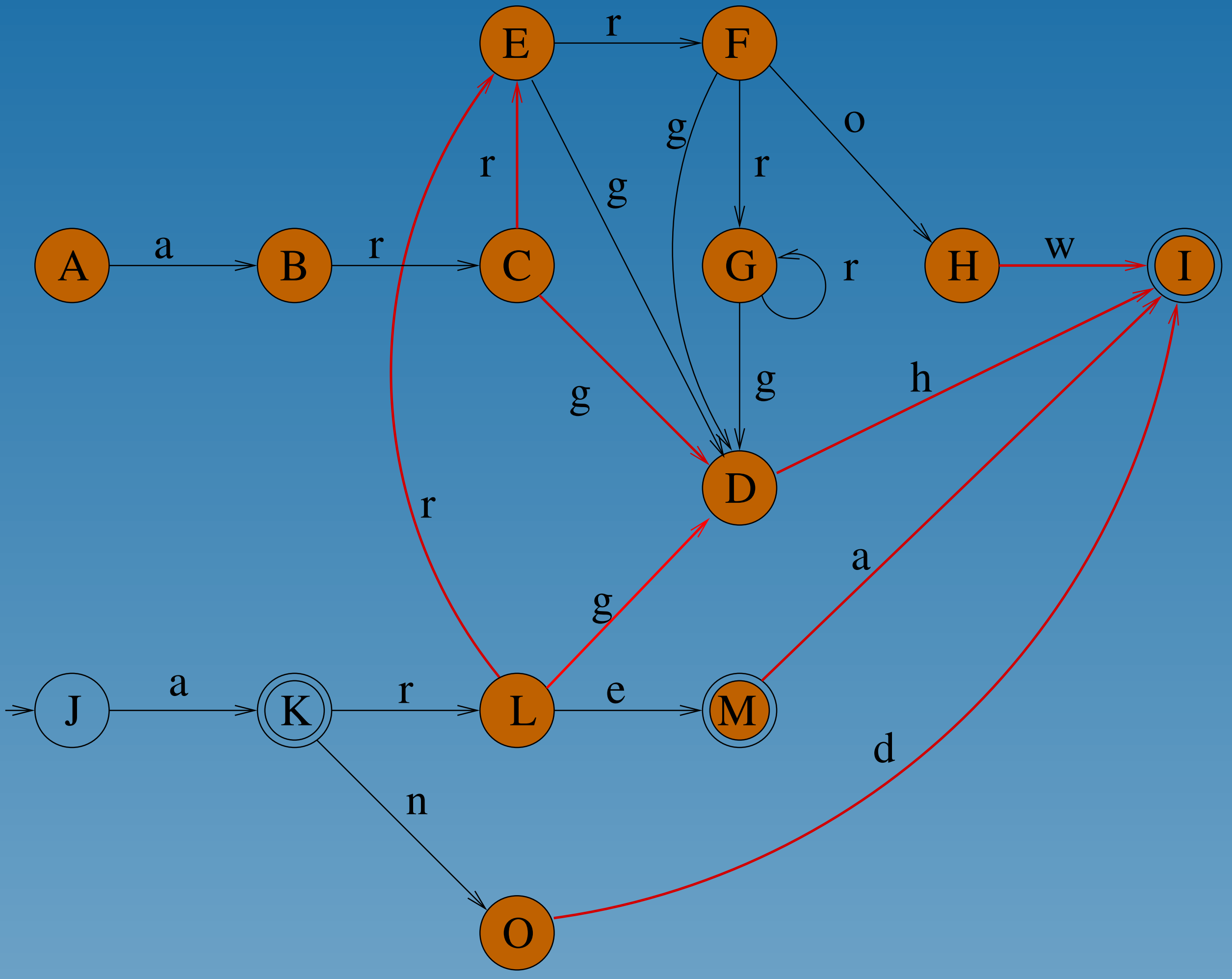
Register = {A,B,C,D,E,F,G,H,I,M,O}

a

Stack = {K,L}

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0$; $i \leftarrow 0$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                    and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
    end while
    $F \leftarrow F \cup \{q\}$;
    return $q$;
end function
```

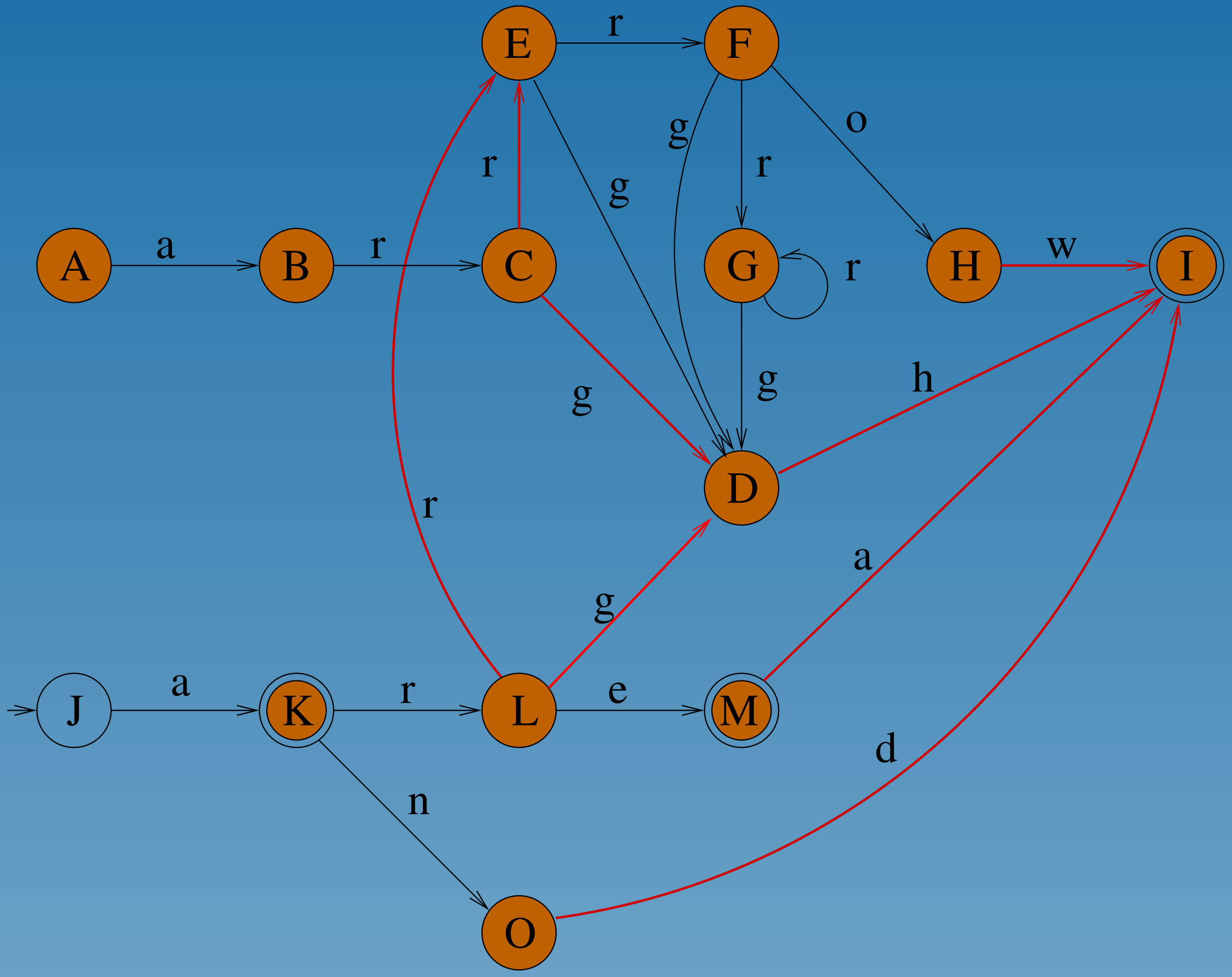


Register = {A,B,C,D,E,F,G,H,I,M,O,L}   a

Stack = {K}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0$; $i \leftarrow 0$;
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                  and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i)$; $i \leftarrow i + 1$;
  end while
  $F \leftarrow F \cup \{q\}$;
  return $q$;
end function
```

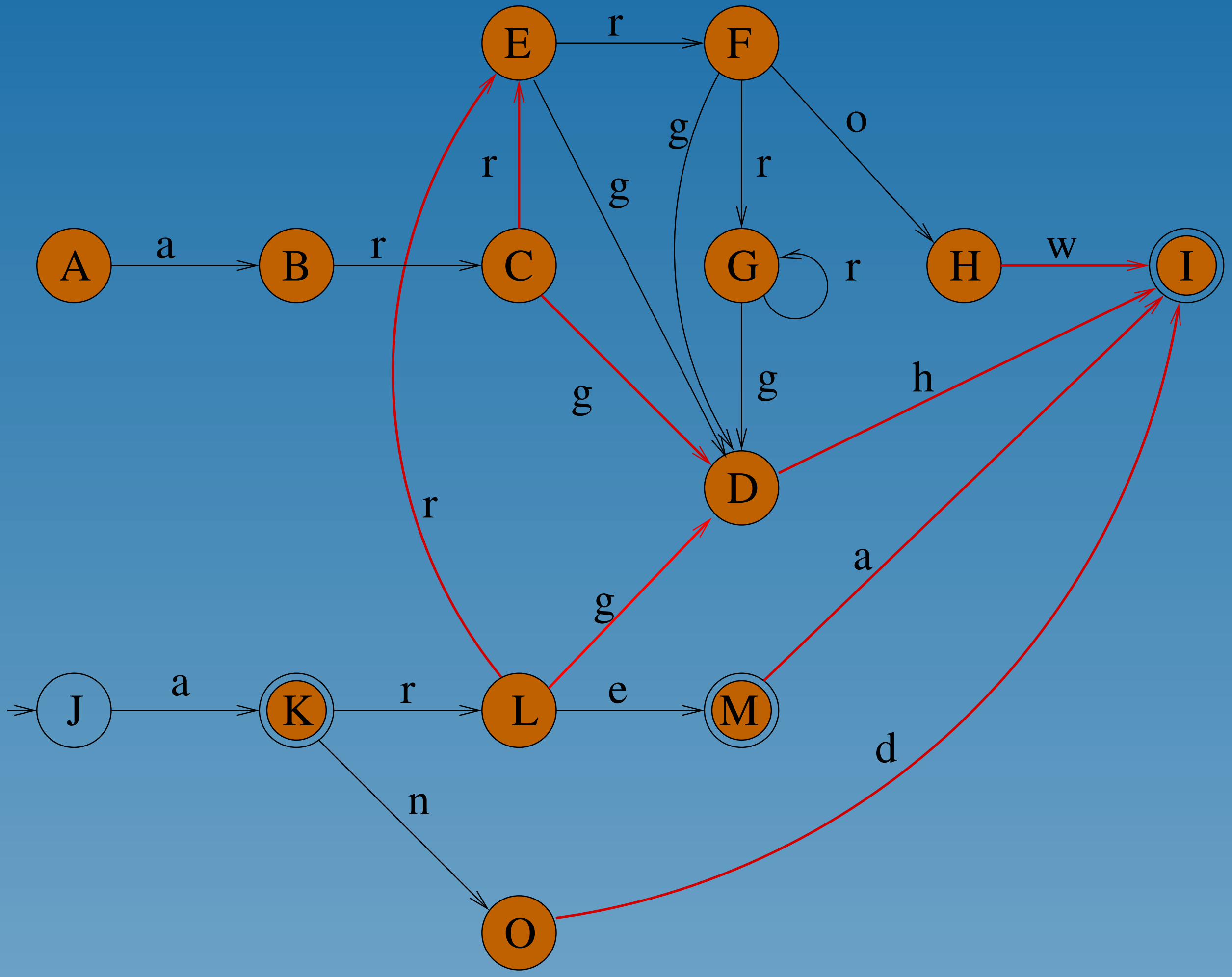


Register = {A,B,C,D,E,F,G,H,I,M,O,L,K}   a

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0; i \leftarrow 0;$
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                    and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
    end while
    $F \leftarrow F \cup \{q\};$
    return $q$;
end function
```

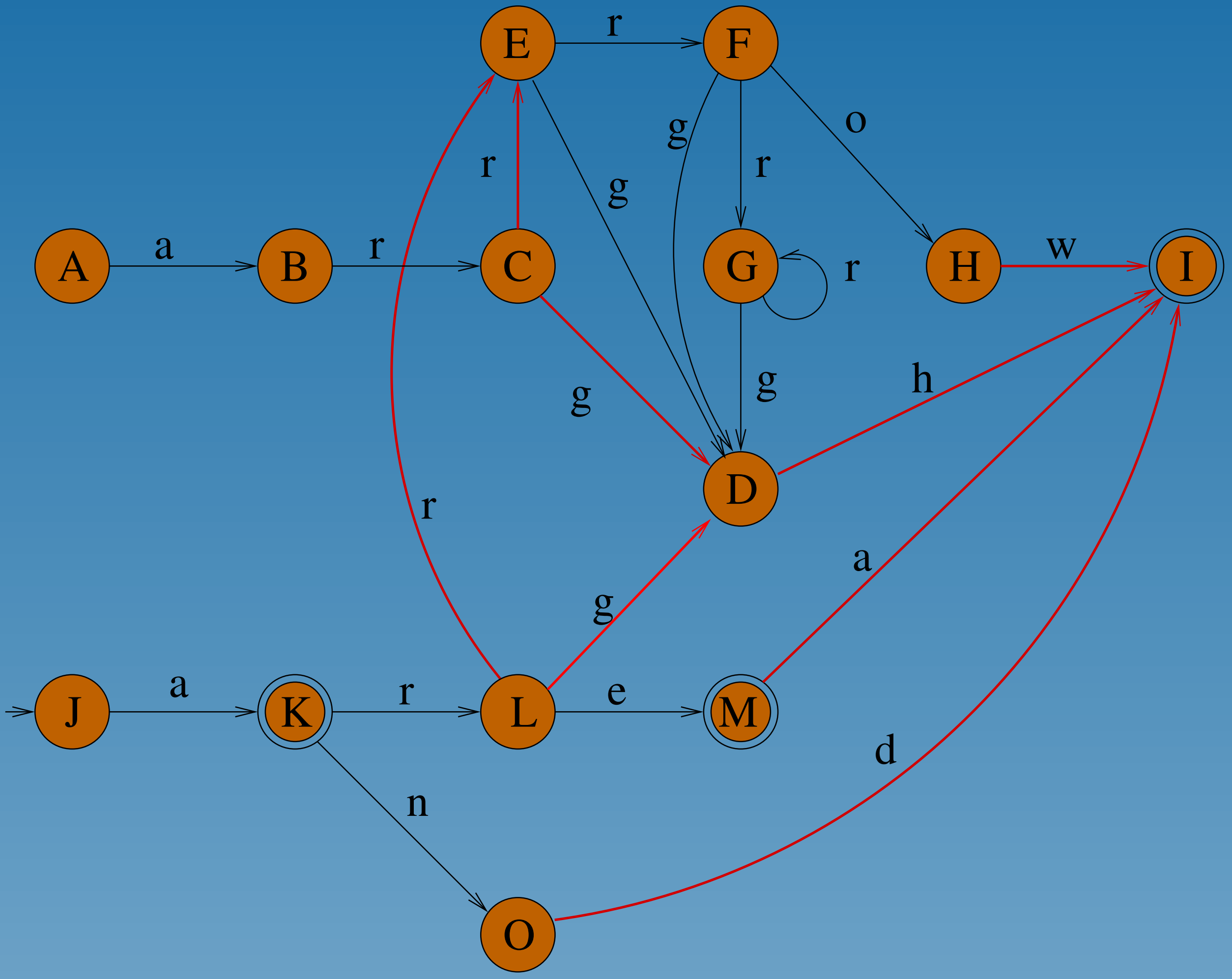


Register = {A,B,C,D,E,F,G,H,I,M,O,L,K}

Stack = {J}

Extension of Watson's Algorithm – Example

```
function add_word(w)
  $q = q_0; i \leftarrow 0;$
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and $fanin(\delta(q, w_i)) < 2$ do
    $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
  end while
  while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
    $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
    $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
  end while
  while $i < |w|$ do
    $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
  end while
  $F \leftarrow F \cup \{q\};$
  return $q;$
end function
```

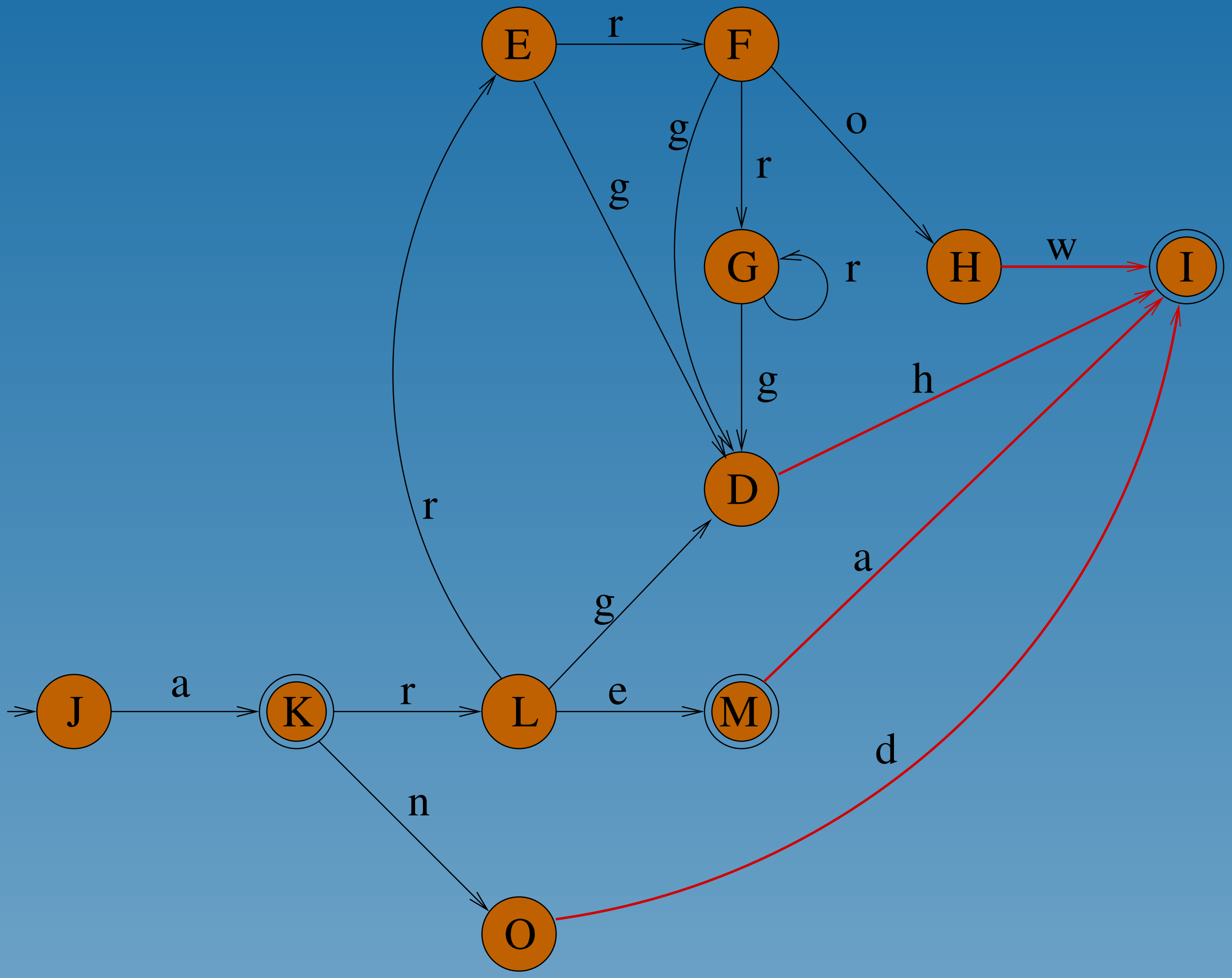


Register = {A,B,C,D,E,F,G,H,I,M,O,L,K,J}

Stack = { }

Extension of Watson's Algorithm – Example

```
function add_word(w)
    $q = q_0; i \leftarrow 0;$
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                 and $fanin(\delta(q, w_i)) < 2$ do
        $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
    end while
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
        $\delta(q, w_i) \leftarrow clone(\delta(q, w_i))$
        $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$
    end while
    while $i < |w|$ do
        $q \leftarrow$ build_state$(q, w_i); i \leftarrow i + 1;$
    end while
    $F \leftarrow F \cup \{q\};$
    return $q;$
end function
```



Register = {D,E,F,G,H,I,M,O,L,K,J}

Stack = { }

Observations

- In the original algorithm, confluence/reentrant states are never encountered
- By cloning the initial state, we make all targets of its transitions confluence/reentrant
- Confluence/reentrant states form a border between the “old” and the “new” part of the automaton
- Cloning confluence/reentrant states restores original conditions for acyclic algorithms

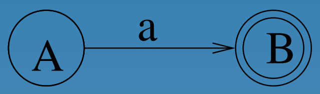
Incremental Algorithm for Sorted Data

```
function add_sorted()
  $R \leftarrow \emptyset$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
    while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if fanout$(q) > 0$ then
      replace_or_register$(q)$;
    end if;
    add_suffix$(q, s)$;
  end while
  replace_or_register$(q_0)$;
end function
```

```
function replace_or_register(s)
  $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
  $c \leftarrow \delta(s, a)$;
  if fanout$(c) > 0$ then
    replace_or_register$(c)$
  end if
  if $\exists_{q \in R} \, q \equiv c$ then
    delete$(c)$; $\delta(s, a) \leftarrow q$;
  else
    $R \leftarrow R \cup \{q\}$;
  end if
end function
```

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```

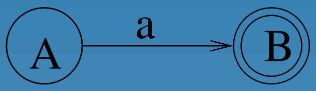


Register = { }

a

Sorted Algorithm – Example

```
function add_sorted()
  $R \leftarrow \emptyset$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
    while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if fanout$(q) > 0$ then
      replace_or_register$(q)$;
    end if;
    add_suffix$(q, s)$;
  end while
  replace_or_register$(q_0)$;
end function
function replace_or_register(s)
  $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
  $c \leftarrow \delta(s, a)$;
  if fanout$(c) > 0$ then
    replace_or_register$(c)$
  end if
  if $\exists_{q \in R}\, q \equiv c$ then
    delete$(c)$; $\delta(s, a) \leftarrow q$;
  else
    $R \leftarrow R \cup \{q\}$;
  end if
end function
```

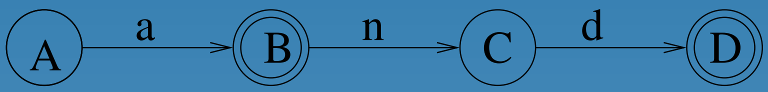


Register = { }

and

Sorted Algorithm – Example

```
function add_sorted()
  $R \leftarrow \emptyset$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
    while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if fanout$(q) > 0$ then
      replace_or_register$(q)$;
    end if;
    add_suffix$(q, s)$;
  end while
  replace_or_register$(q_0)$;
end function
function replace_or_register(s)
  $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
  $c \leftarrow \delta(s, a)$;
  if fanout$(c) > 0$ then
    replace_or_register$(c)$
  end if
  if $\exists_{q \in R}\, q \equiv c$ then
    delete$(c)$; $\delta(s, a) \leftarrow q$;
  else
    $R \leftarrow R \cup \{q\}$;
  end if
end function
```

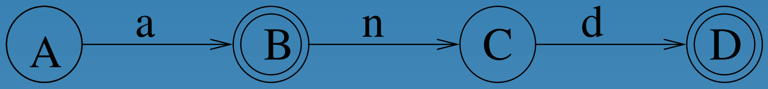


Register = { }

and

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```

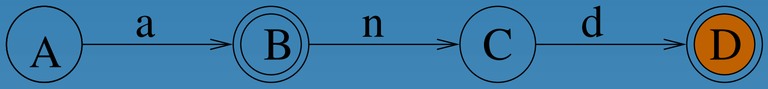


Register = { }

are

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```

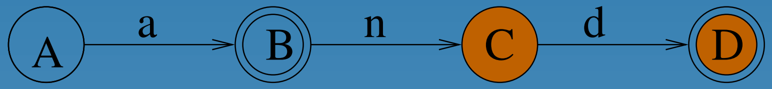


Register = {D}

are

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```

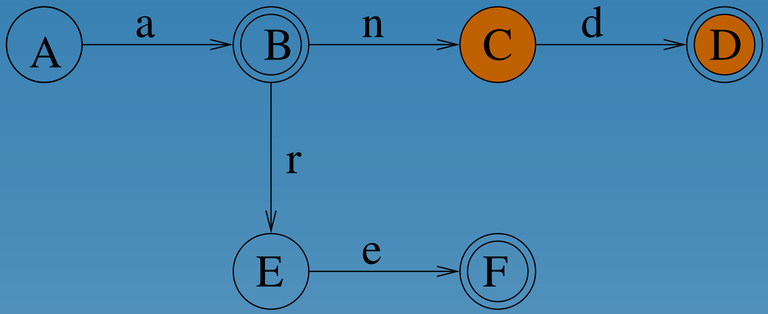


Register = {D,C}

are

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```

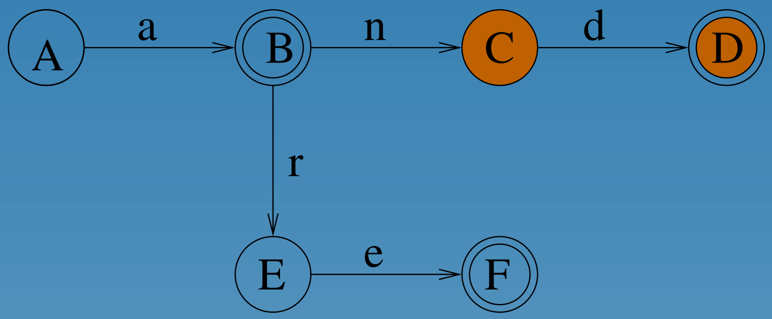


Register = {D,C}

are

Sorted Algorithm – Example

```
function add_sorted()
  $R \leftarrow \emptyset$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
    while $i \leq |w|$ and $\delta(q, w_i) \neq \bot$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if fanout$(q) > 0$ then
      replace_or_register$(q)$;
    end if;
    add_suffix$(q, s)$;
  end while
  replace_or_register$(q_0)$;
end function
function replace_or_register(s)
  $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \bot\}$;
  $c \leftarrow \delta(s, a)$;
  if fanout$(c) > 0$ then
    replace_or_register$(c)$
  end if
  if $\exists_{q \in R}\, q \equiv c$ then
    delete$(c)$; $\delta(s, a) \leftarrow q$;
  else
    $R \leftarrow R \cup \{q\}$;
  end if
end function
```

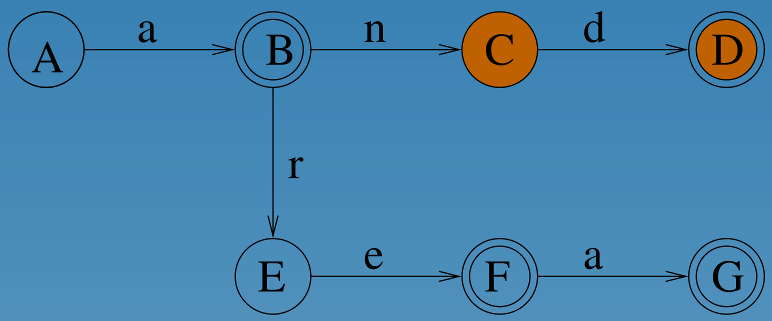


Register = {D,C}

area

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```

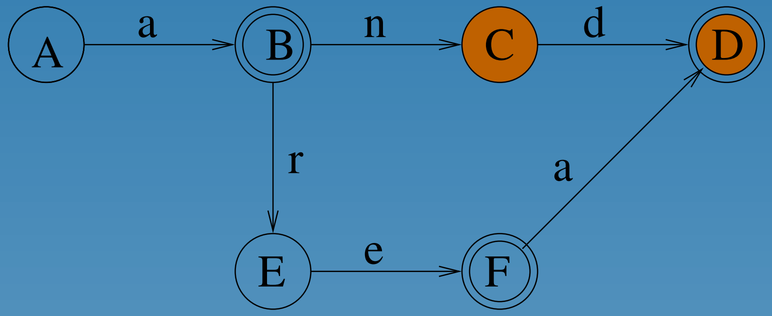


Register = {D,C}

area

Sorted Algorithm – Example

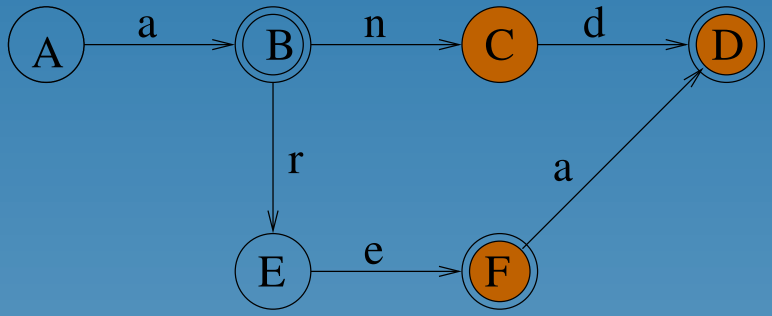
```
function add_sorted()
  $R \leftarrow \emptyset$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
    while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if fanout$(q) > 0$ then
      replace_or_register$(q)$;
    end if;
    add_suffix$(q, s)$;
  end while
  replace_or_register$(q_0)$;
end function
function replace_or_register(s)
  $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
  $c \leftarrow \delta(s, a)$;
  if fanout$(c) > 0$ then
    replace_or_register$(c)$
  end if
  if $\exists_{q \in R}\, q \equiv c$ then
    delete$(c)$; $\delta(s, a) \leftarrow q$;
  else
    $R \leftarrow R \cup \{q\}$;
  end if
end function
```



Register = {D,C}

Sorted Algorithm – Example

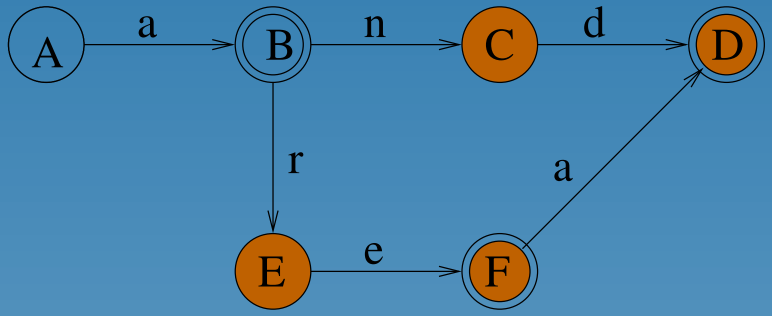
```
function add_sorted()
   $R \leftarrow \emptyset$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
      while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if fanout$(q) > 0$ then
         replace_or_register$(q)$;
      end if;
      add_suffix$(q, s)$;
   end while
   replace_or_register$(q_0)$;
end function
function replace_or_register(s)
   $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
   $c \leftarrow \delta(s, a)$;
   if fanout$(c) > 0$ then
      replace_or_register$(c)$
   end if
   if $\exists_{q \in R}\, q \equiv c$ then
      delete$(c)$; $\delta(s, a) \leftarrow q$;
   else
      $R \leftarrow R \cup \{q\}$;
   end if
end function
```



Register = {D,C,F}

Sorted Algorithm – Example

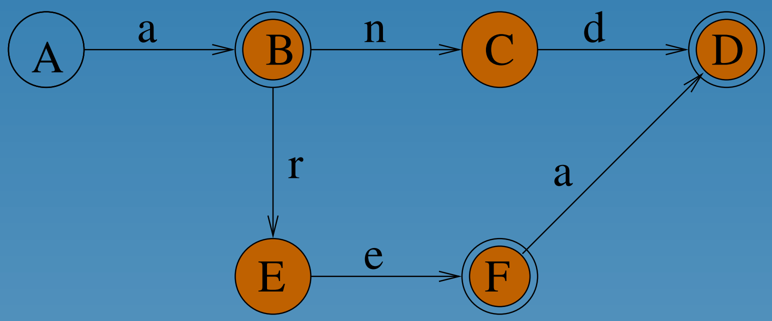
```
function add_sorted()
  $R \leftarrow \emptyset$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
    while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if fanout$(q) > 0$ then
      replace_or_register$(q)$;
    end if;
    add_suffix$(q, s)$;
  end while
  replace_or_register$(q_0)$;
end function
function replace_or_register(s)
  $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
  $c \leftarrow \delta(s, a)$;
  if fanout$(c) > 0$ then
    replace_or_register$(c)$
  end if
  if $\exists_{q \in R}\, q \equiv c$ then
    delete$(c)$; $\delta(s, a) \leftarrow q$;
  else
    $R \leftarrow R \cup \{q\}$;
  end if
end function
```



Register = {D,C,F,E}

Sorted Algorithm – Example

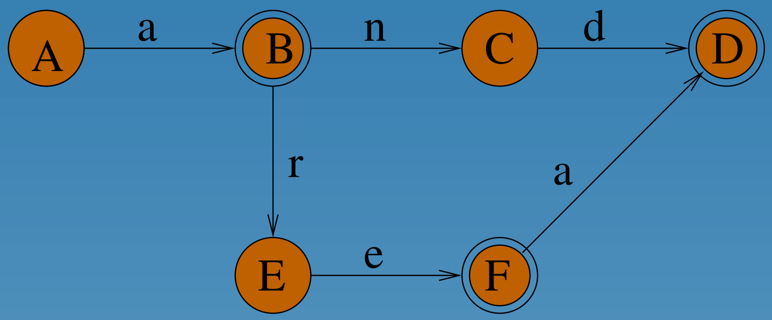
```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \perp$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \perp\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```


$$\begin{tikzcd}
A \arrow[r, "a"] & B \arrow[r, "n"] \arrow[d, "r"] & C \arrow[r, "d"] & D \\
 & E \arrow[r, "e"] & F \arrow[ur, "a"] &
\end{tikzcd}$$

Register = {D,C,F,E,B}

Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow \emptyset$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow q_0$;
        while $i \leq |w|$ and $\delta(q, w_i) \neq \bot$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if fanout$(q) > 0$ then
            replace_or_register$(q)$;
        end if;
        add_suffix$(q, s)$;
    end while
    replace_or_register$(q_0)$;
end function
function replace_or_register(s)
    $a \leftarrow \max\{a \in \Sigma | \delta(s, a) \neq \bot\}$;
    $c \leftarrow \delta(s, a)$;
    if fanout$(c) > 0$ then
        replace_or_register$(c)$
    end if
    if $\exists_{q \in R}\, q \equiv c$ then
        delete$(c)$; $\delta(s, a) \leftarrow q$;
    else
        $R \leftarrow R \cup \{q\}$;
    end if
end function
```



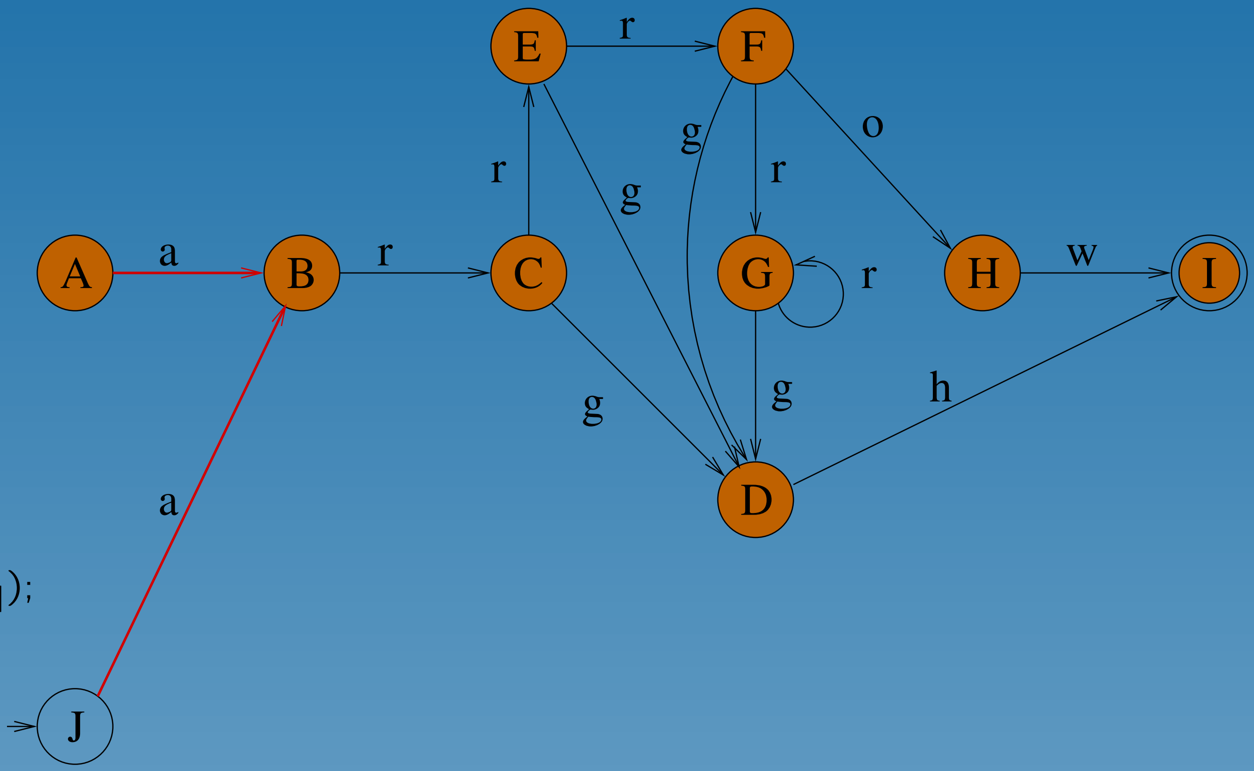
Register = {D,C,F,E,B,A}

Extension of the Sorted Algorithm

```
function add_sorted()
  $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;          $\longleftarrow$ clone initial state
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
    $j \leftarrow |w \wedge w'|$;
    while $i < |w|$ and $\delta(q, w_i) \neq \bot$ and fanin($\delta(q, w_i)$) $< 2$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    while $i < |w|$ and $\delta(q, w_i) \neq \bot$ do
      $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $|w'| \geq j$ then          $\longleftarrow$ used to be fanout($q$) $> 0$
      replace_or_register($\delta(r, w_{1 \ldots j-1}, w'_{j \ldots |w'|}$);
    end if;
    add_suffix($q, w_{i \ldots |w|}$);
    $w' \leftarrow w$;          $\longleftarrow$ remember previous word
  end while;
  replace_or_register($r, w'$);          $\longleftarrow$ follow letters instead of last transition
  if $r \neq q_0$ then
    delete_branch($q_0$);          $\longleftarrow$ delete unreachable states
  end if
end function
```

Extended Sorted Algorithm – Example

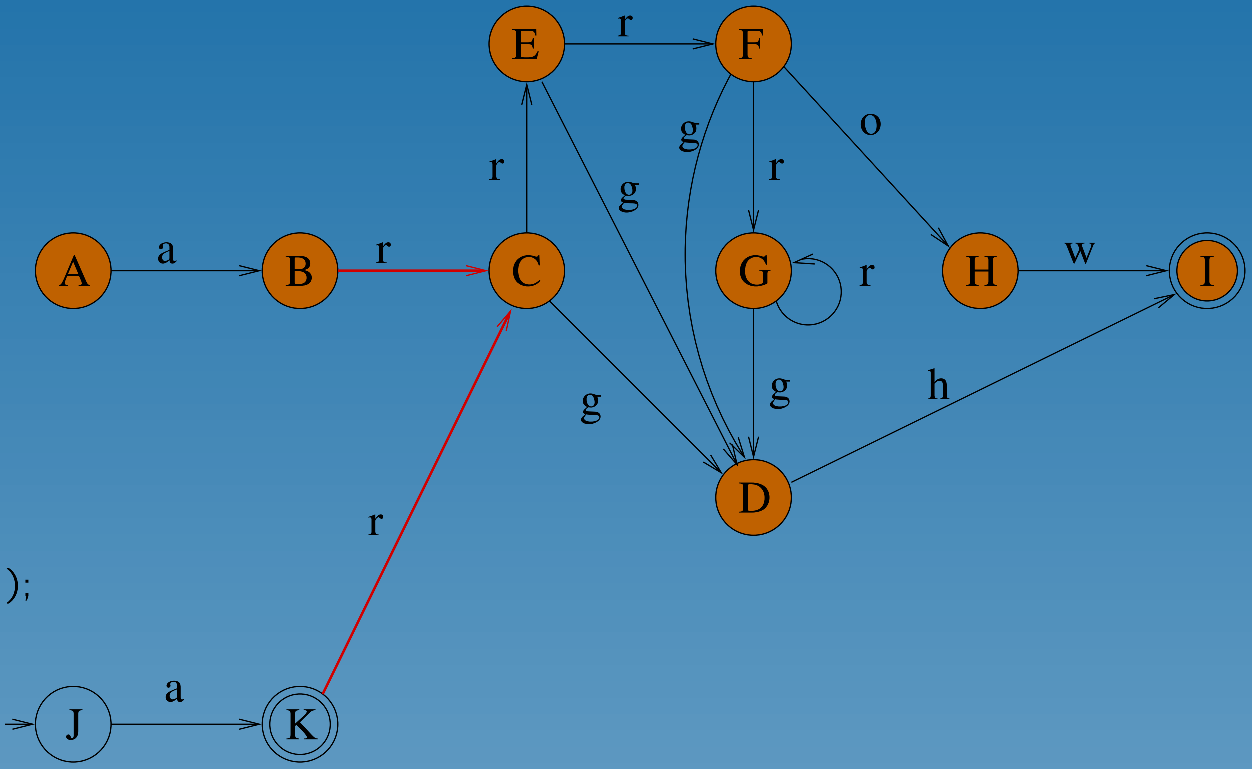
```
function add_sorted()
    $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
        $j \leftarrow |w \wedge w'|$;
        while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and fanin($\delta(q, w_i)$) $< 2$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
            $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $|w'| \geq j$ then
            replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
        end if;
        add_suffix($q, w_{i \ldots |w|}$);
        $w' \leftarrow w$;
    end while;
    replace_or_register($q_0$);
    if $r \neq q_0$ then
        delete_branch($q_0$);
    end if
end function
```



Register = {A,B,C,D,E,F,G,H,I}

Extended Sorted Algorithm – Example

```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($\delta(r, w_{1 \ldots j-1}), w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($w, w'$);
   if $r \neq q_0$ then
      delete_branch($q_0$);
   end if
end function
```

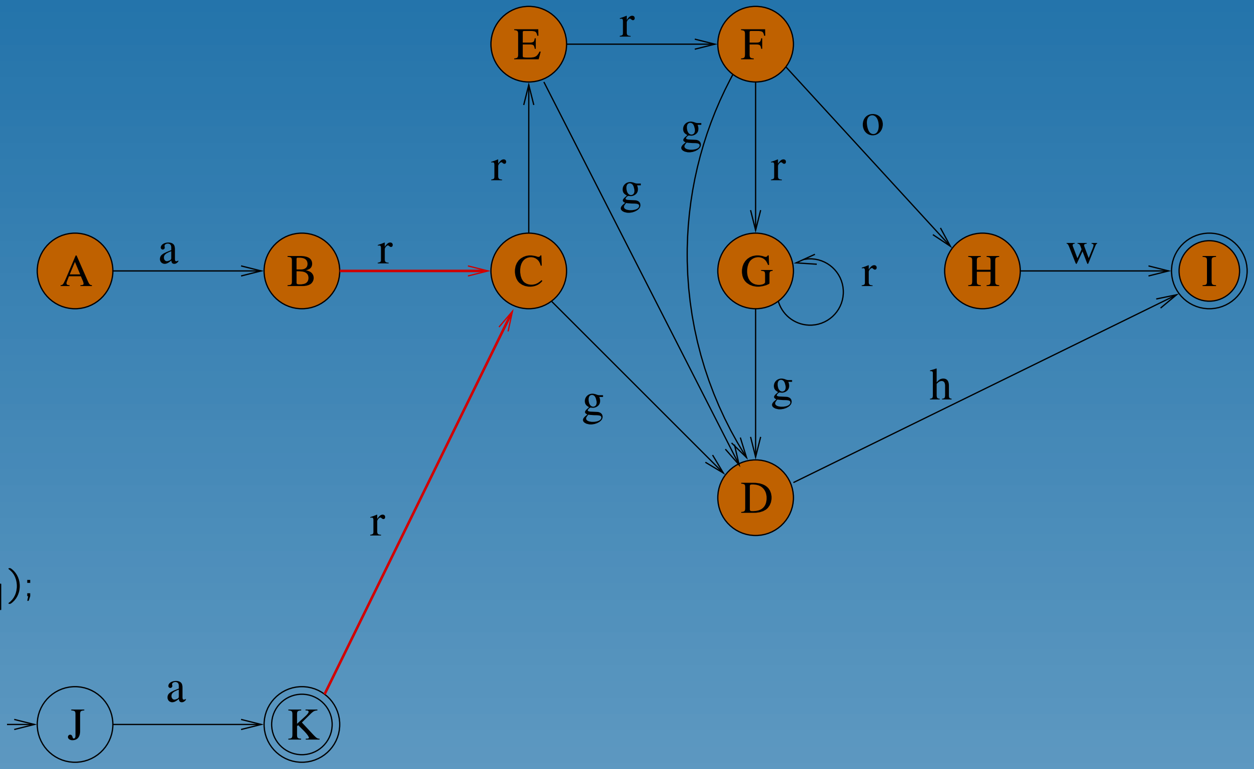


Register = {A,B,C,D,E,F,G,H,I}

a

Extended Sorted Algorithm – Example

```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($q, w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($r, w'$);
   if $r \neq q_0$ then
      delete_branch($q_0$);
   end if
end function
```

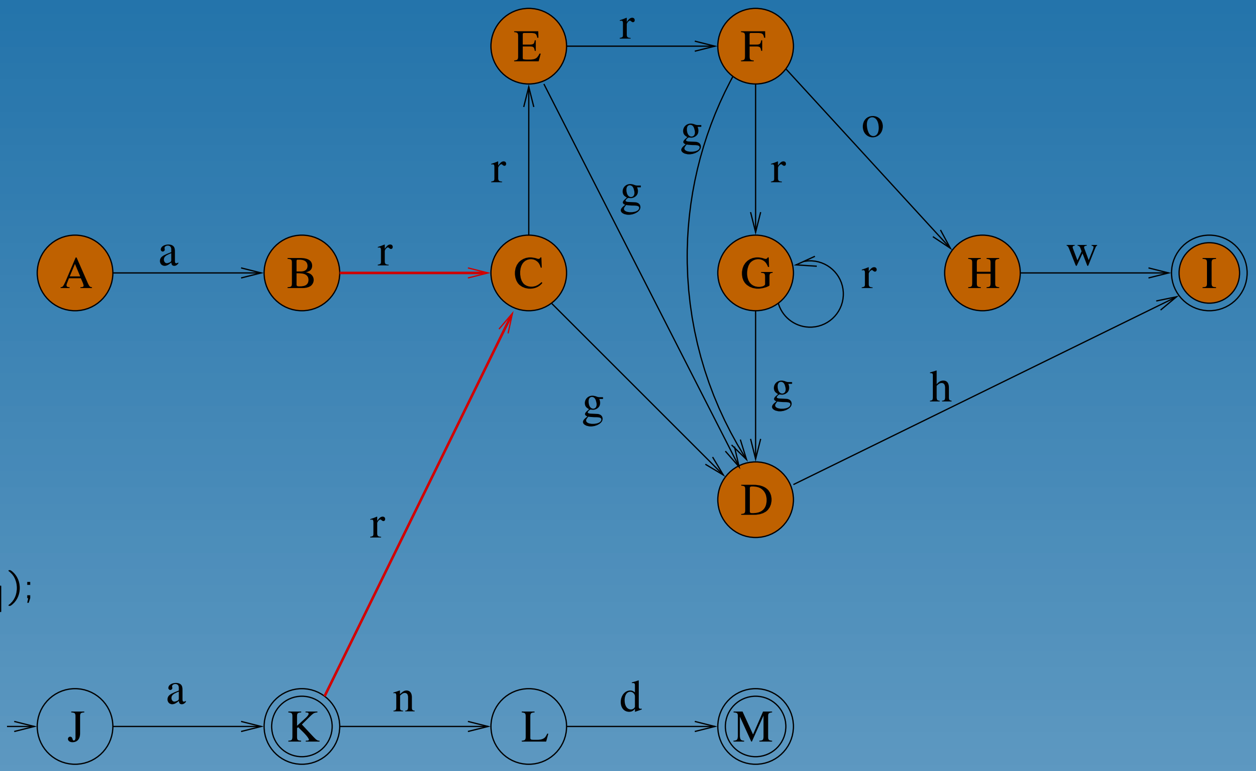


Register = {A,B,C,D,E,F,G,H,I}

and

Extended Sorted Algorithm – Example

```
function add_sorted()
   $R \leftarrow Q; r \leftarrow \text{clone}(q_0); w' \leftarrow \epsilon;$
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1; q \leftarrow r;$
      $j \leftarrow |w \wedge w'|;$
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin$(\delta(q, w_i)) < 2$ do
         $q \leftarrow \delta(q, w_i); i \leftarrow i + 1;$
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow \text{clone}(\delta(q, w_i));$
         $q \leftarrow \delta(q, w_i); i \leftarrow i + 1;$
      end while;
      if $|w'| \geq j$ then
         replace_or_register$(\delta(r, w_{1\ldots j-1}), w_{j\ldots|w|});$
      end if;
      add_suffix$(q, w_{i\ldots|w|});$
      $w' \leftarrow w;$
   end while;
   replace_or_register$(r, w');$
   if $r \neq q_0$ then
      delete_branch$(q_0);$
   end if
end function
```

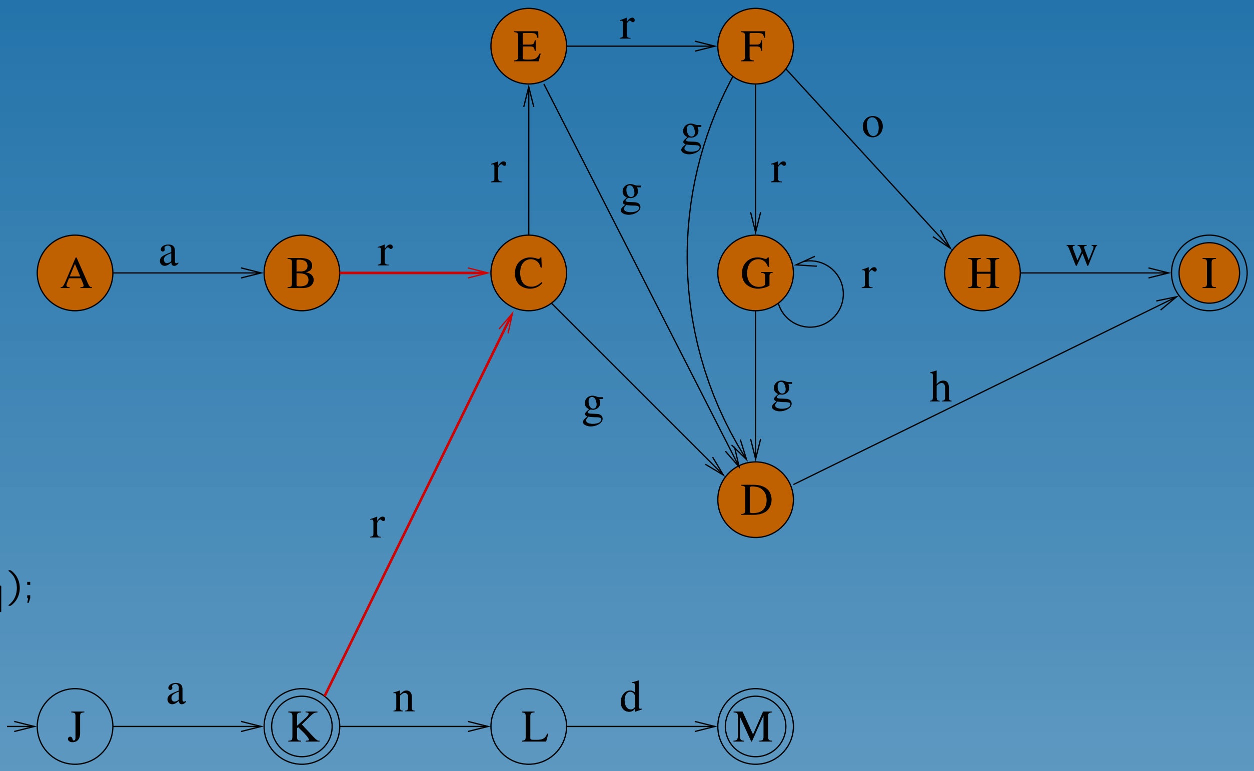


Register = {A,B,C,D,E,F,G,H,I}

and

Extended Sorted Algorithm – Example

```
function add_sorted()
    $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
    while not eof do
        $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
        $j \leftarrow |w \wedge w'|$;
        while $i < |w|$ and $\delta(q, w_i) \neq \perp$
                and fanin($\delta(q, w_i)$) $< 2$ do
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
            $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
            $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $|w'| \geq j$ then
            replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
        end if;
        add_suffix($q, w_{i \ldots |w|}$);
        $w' \leftarrow w$;
    end while;
    replace_or_register($r, w'$);
    if $r \neq q_0$ then
        delete_branch($q_0$);
    end if
end function
```

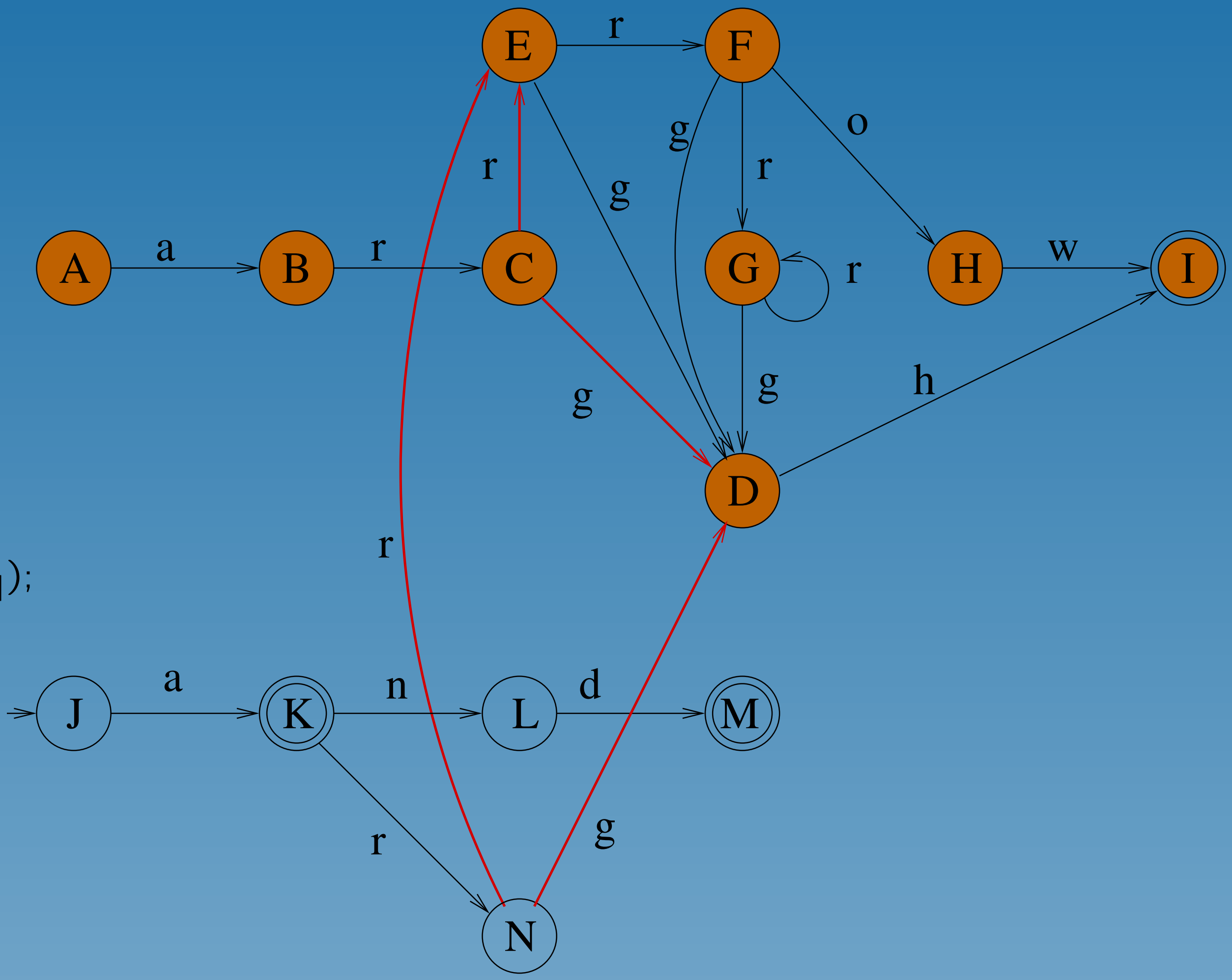


Register = {A,B,C,D,E,F,G,H,I}

are

Extended Sorted Algorithm – Example

```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1})$, $w_{j \ldots |w|}$);
      end if;
      add_suffix($q$, $w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($r$, $w'$);
   if $r \neq q_0$ then
      delete_branch($q_0$);
   end if
end function
```

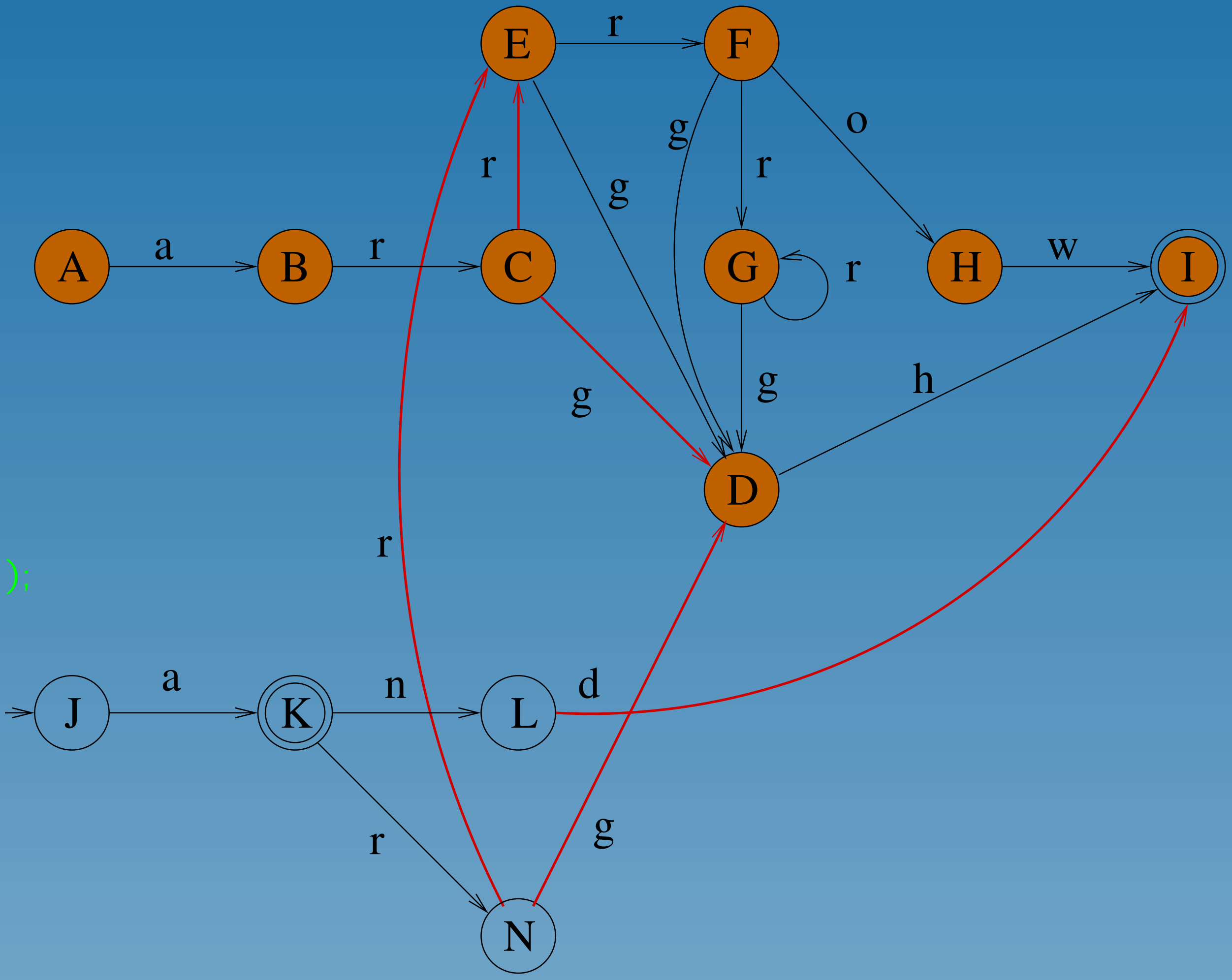


Register = {A,B,C,D,E,F,G,H,I}

are

Extended Sorted Algorithm – Example

```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($q, w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($r, w'$);
   if $r \neq q_0$ then
      delete_branch($q_0$);
   end if
end function
```

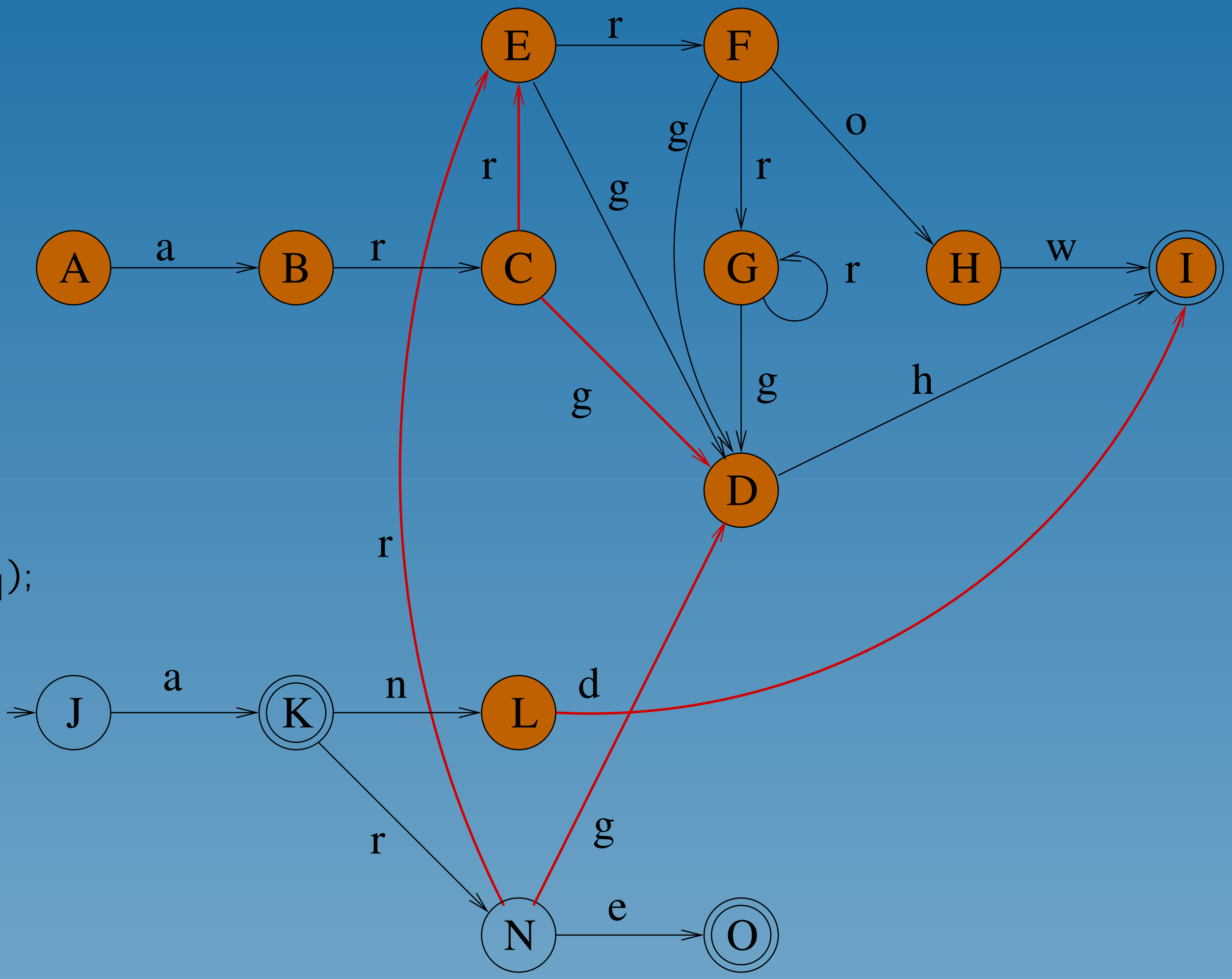


Register = {A,B,C,D,E,F,G,H,I}

are

Extended Sorted Algorithm – Example

```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($q, w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($r, w'$);
   if $r \neq q_0$ then
      delete_branch($q_0$);
   end if
end function
```

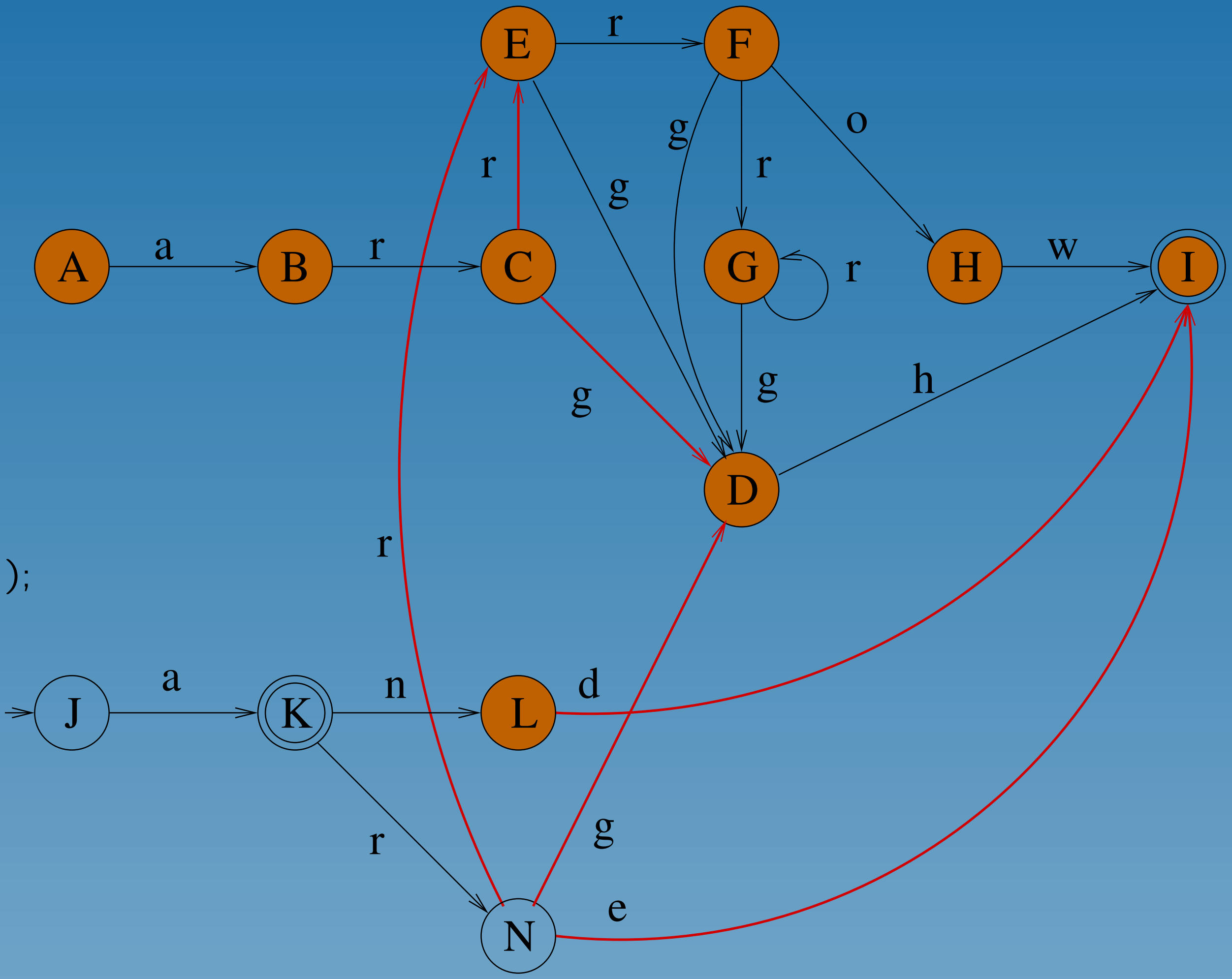


Register = {A,B,C,D,E,F,G,H,I,L}

are

Extended Sorted Algorithm – Example

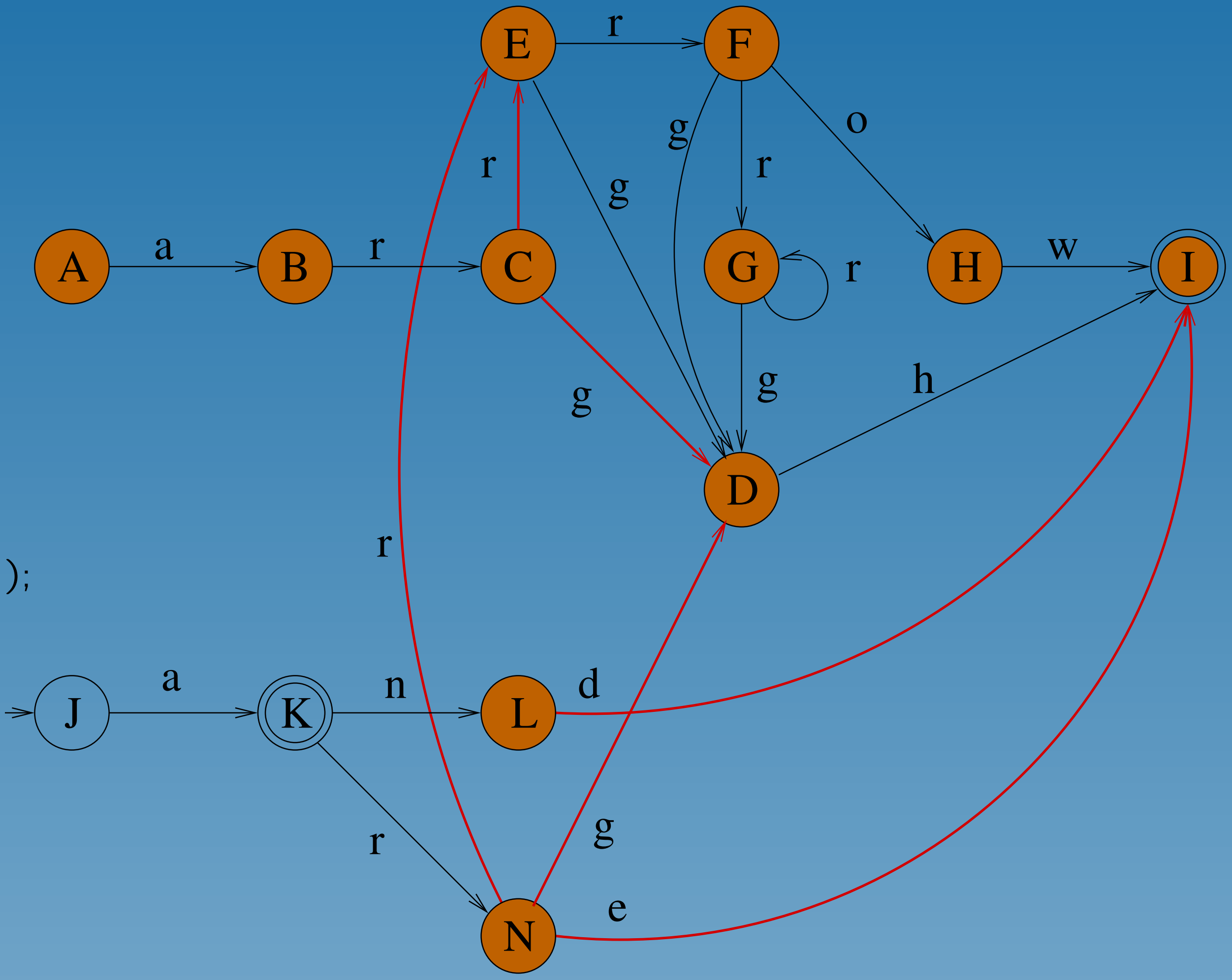
```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($q, w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($q_0$);
   if $r \neq q_0$ then
      delete_branch($r, w'$);
   end if
end function
```



Register = {A,B,C,D,E,F,G,H,I,L}

Extended Sorted Algorithm – Example

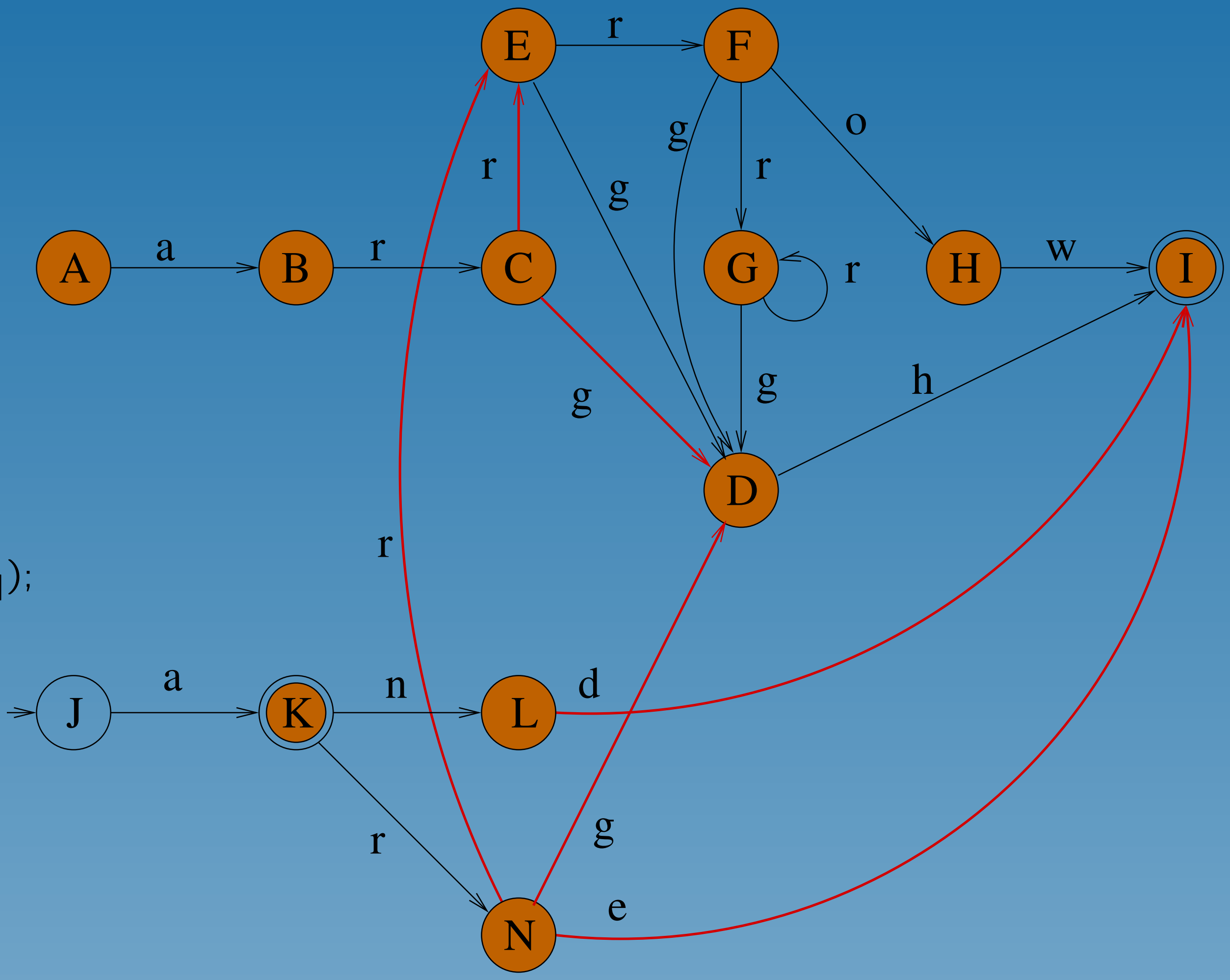
```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($q, w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($q_0$);
   if $r \neq q_0$ then
      delete_branch($r, w'$);
   end if
end function
```



Register = {A,B,C,D,E,F,G,H,I,L,N}

Extended Sorted Algorithm – Example

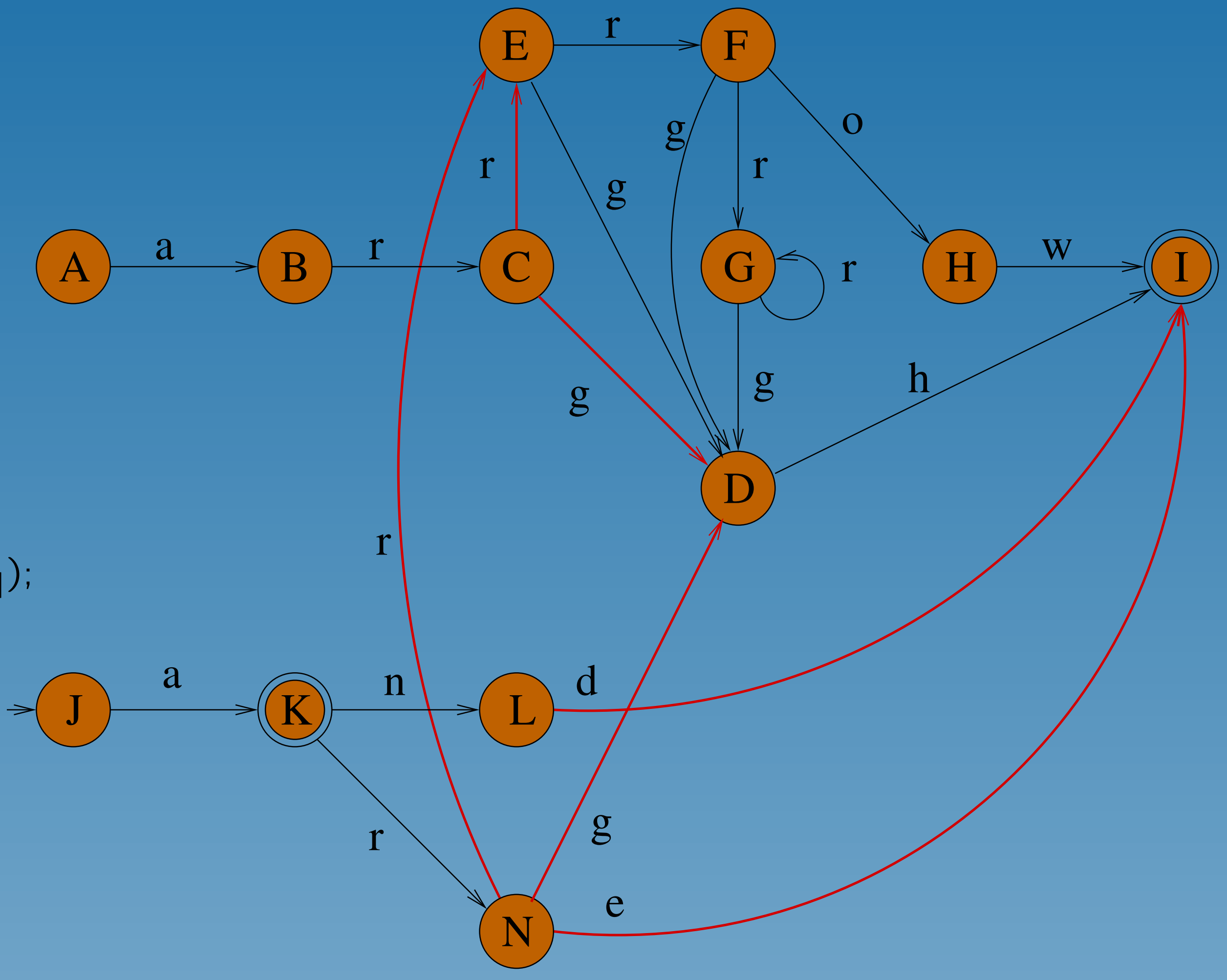
```
function add_sorted()
  $R \leftarrow Q; r \leftarrow$ clone$(q_0); w' \leftarrow \epsilon$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1; q \leftarrow r$;
    $j \leftarrow |w \wedge w'|$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
        and fanin$(\delta(q, w_i)) < 2$ do
      $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$;
    end while;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
      $\delta(q, w_i) \leftarrow$ clone$(\delta(q, w_i))$;
      $q \leftarrow \delta(q, w_i); i \leftarrow i + 1$;
    end while;
    if $|w'| \geq j$ then
      replace_or_register$(\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|})$;
    end if;
    add_suffix$(q, w_{i \ldots |w|})$;
    $w' \leftarrow w$;
  end while;
  replace_or_register$(q_0)$;
  if $r \neq q_0$ then
    delete_branch$(r, w')$;
  end if
end function
```



Register = {A,B,C,D,E,F,G,H,I,L,N,K}

Extended Sorted Algorithm – Example

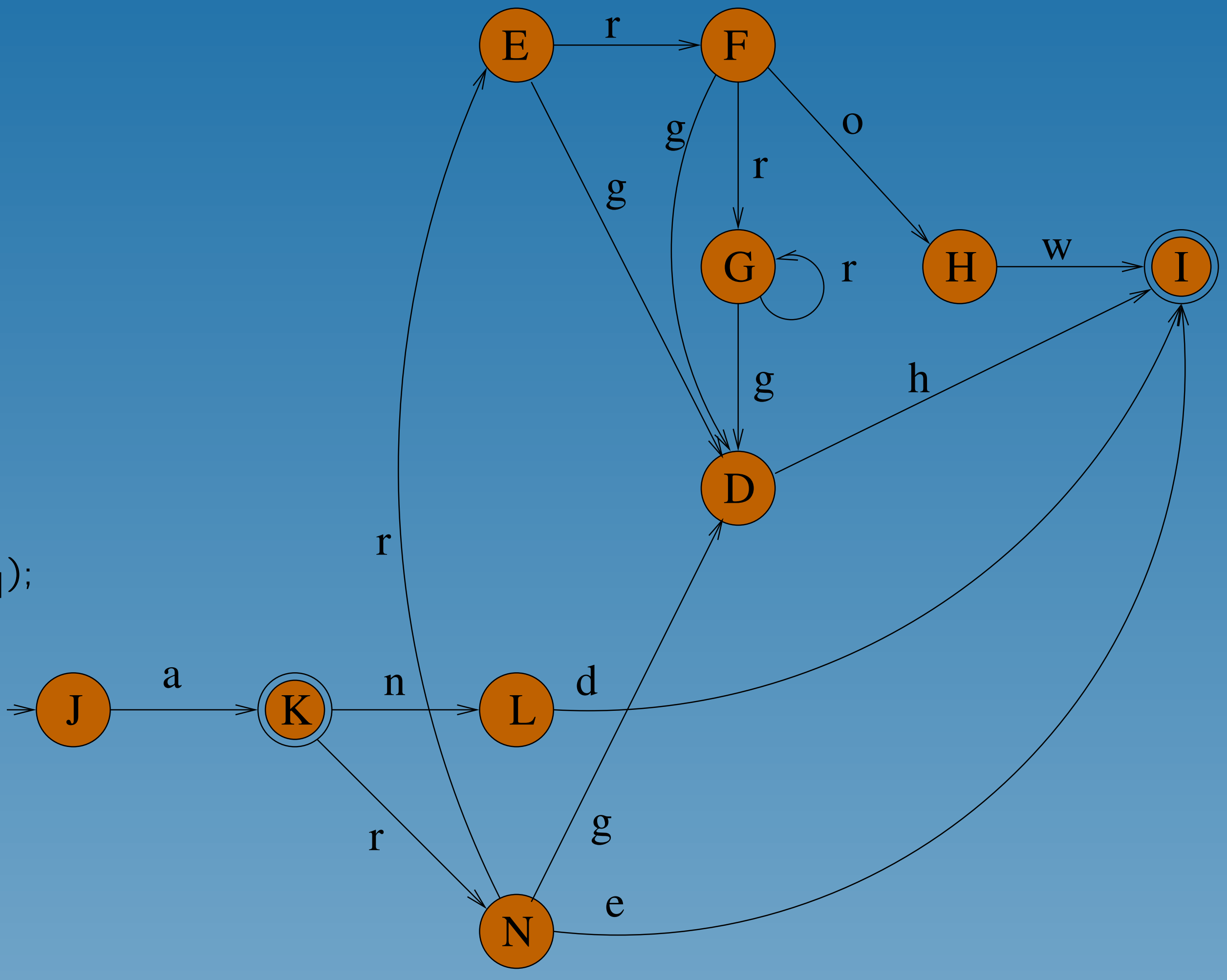
```
function add_sorted()
   $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
   while not eof do
      $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
      $j \leftarrow |w \wedge w'|$;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$
            and fanin($\delta(q, w_i)$) $< 2$ do
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
         $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
         $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $|w'| \geq j$ then
         replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
      end if;
      add_suffix($q, w_{i \ldots |w|}$);
      $w' \leftarrow w$;
   end while;
   replace_or_register($q_0$);
   if $r \neq q_0$ then
      delete_branch($r, w'$);
   end if
end function
```



Register = {A,B,C,D,E,F,G,H,I,L,N,K,J}

Extended Sorted Algorithm – Example

```
function add_sorted()
  $R \leftarrow Q$; $r \leftarrow$ clone($q_0$); $w' \leftarrow \epsilon$;
  while not eof do
    $w \leftarrow$ next_word; $i \leftarrow 1$; $q \leftarrow r$;
    $j \leftarrow |w \wedge w'|$;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$
        and fanin($\delta(q, w_i)$) $< 2$ do
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    while $i < |w|$ and $\delta(q, w_i) \neq \perp$ do
      $\delta(q, w_i) \leftarrow$ clone($\delta(q, w_i)$);
      $q \leftarrow \delta(q, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $|w'| \geq j$ then
      replace_or_register($\delta(r, w_{1 \ldots j-1}), w_{j \ldots |w|}$);
    end if;
    add_suffix($q, w_{i \ldots |w|}$);
    $w' \leftarrow w$;
  end while;
  replace_or_register($q_0$);
  if $r \neq q_0$ then
    delete_branch($r, w'$);
  end if
end function
```



Register = {D,E,F,G,H,I,L,N,K,J}

Observations

- In the original algorithm, confluence/reentrant states are never encountered
- By cloning the initial state, we make all targets of its transitions confluence/reentrant
- Confluence/reentrant states form a border between the "old" and the "new" part of the automaton
- Cloning confluence/reentrant states restores original conditions for acyclic algorithms

Conclusions

- It is possible to extend additional two construction algorithms for acyclic automata to the cyclic case in a similar way to Carrasco and Forcada's extension of the unsorted data incremental construction algorithm

- The new algorithms are faster than the one proposed by Carrasco and Forcada as they do not have to reprocess the same states over and over again

- The new algorithms require data to be sorted, but data may come either sorted or in sorted chunks, and in case of Watson's algorithm, sorting comes for free (distribution instead of sorting)