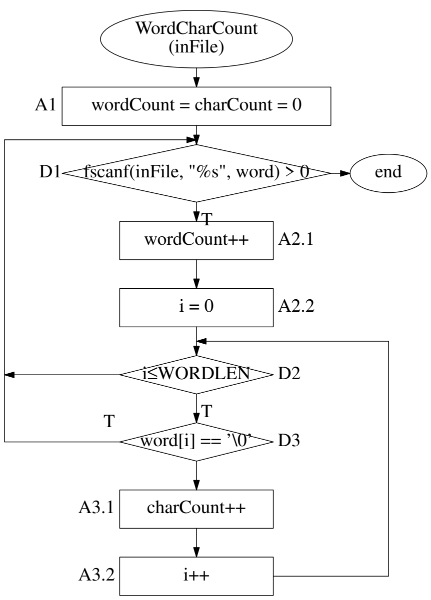
FINITE-STATE MODEL AND FLOWCHARTS

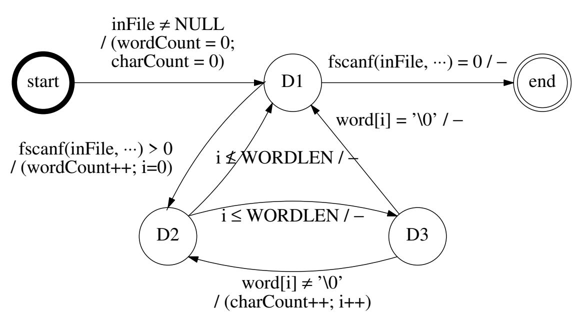
States:
- Each state s_i represents a "phase" in the computation being modeled; the number of states is *finite*.
- There are two special states: start-state and end-state.
- The complete *state-of-computation* = state + current values of all variables.

Transitions:
- Each transition $t(s_i, c_{ij}, a_{ij}, s_j)$ represents an applicable condition-action-newState triplet at a state s_i.
- The unique transition from the start-state represents the conditions on input parameters and the initializations actions (if any).
- There is no transition from the end-state.
- Each state belongs to a path from the start-state to the end-state.

Deterministic:
- For each state s_i, *at most* one of the transitions at s_i is applicable (i.e., its condition-part is satisfied) for any values of variables.

Complete:
- For each state s_i (except possibly start-state), *at least* one of the transitions at s_i is applicable for any values of variables.





The start-state is often merged
with its uniue next-state.

The flowchart of WordCharCount-function and its FSM. The transition (D2, D1) never occurs because before "$i \nleq$ WORDLEN" becomes true (D3, D1) succeeds; (D2, D3) always succeeds.

- FSMs are more powerful than DFDs because they can represent branching. Cycles (even loops) occur frequently in FSM.
- FSMs are more detailed than DFDs and less detailed than flowcharts.

Decision-To-Decision Path (dd-path):

A path starting at a branch-node or the start-node and ending in a branch-node or the end-node, without going through any other branch-node.

Facts: For FSM obtained from a flowchart,

(1) Each dd-path gives a transition.

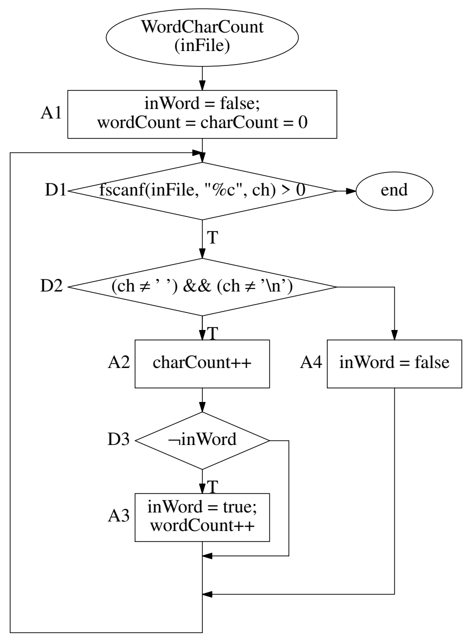
(2) Each state, save the start-state and the end-state, has exactly two transitions.

#(transitions) = 1 + 2×#(branch-nodes).

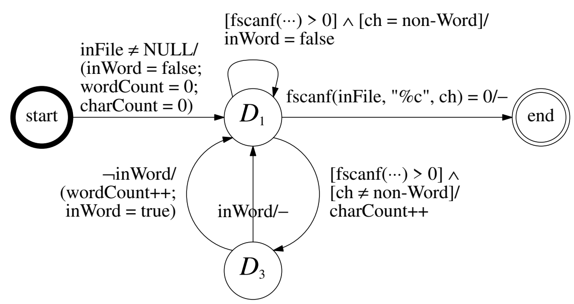
A DIFFERENT ALGORITHM FOR WORDCOUNT-CHARCOUNT AND ITS FSM

This algorithm is suitable when the words in the input file do not have a predefined bound on their lengths.

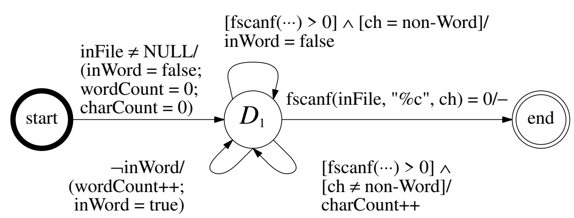
```
void NewWordCharCount(FILE *inFile)
{int inWord=FALSE; //= TRUE after first character of word is seen
 char ch;
 wordCount = charCount = 0;
 while (fscanf(inFile, "%c", &ch) > 0) {
       if ((ch != ' ') && (ch != '\n')) { //ch ≠ non-word symbol
          charCount++;
          if (!inWord) {
             inWord = TRUE; wordCount++;}
       } else inWord = FALSE;
 }
}
```



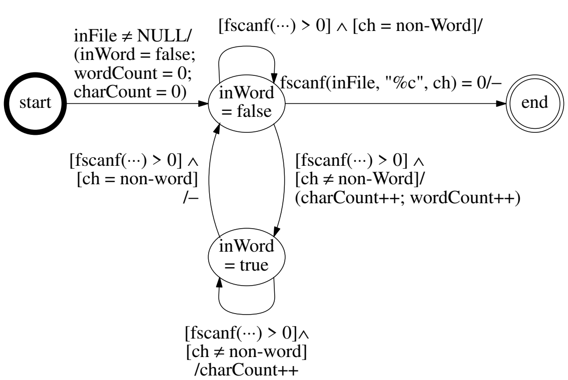
(i) The flowchart of a new WordCharCount function.



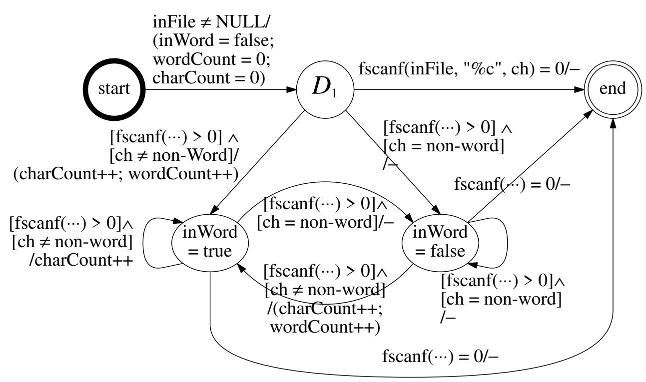
(ii) After elimination of the transition for the *dd*-path (D_1, D_2) which has null-action.



(iii) After elimination of the transition (D_1, D_3) which precede both the transitions from D_3 to D_1.

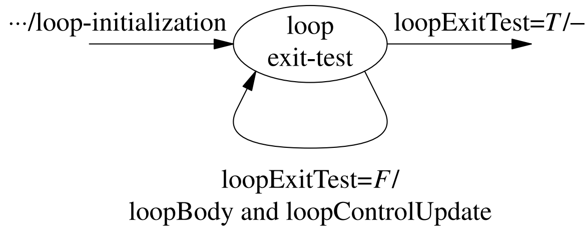


(iv) The result of splitting D_1 in (iii) into two states based on values of "inword".



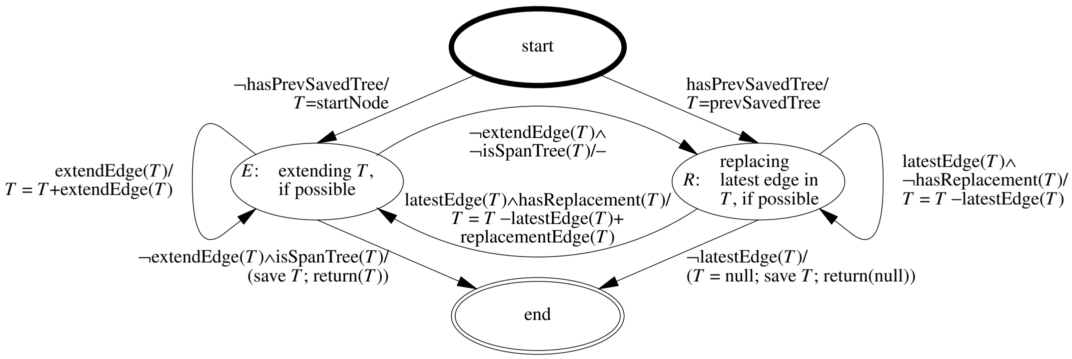
(v) An alternate FSM obtained by replacing D_3 in (iii) by two states based on values of "inWord".

CONTRIBUTION TO FSM FROM A LOOP



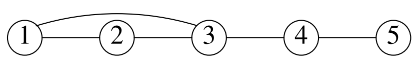
AN FSM FOR GENERATING SPANNING TREES IN A CONNECTED GRAPH

- Each spanning tree is built by successively adding edges in a depth-first manner.
- Each call generates a new spanning tree, if any, by deleting the most recent edge and then extending the partial tree in a depth-first fashion by addition of zero or more edges; there may be many 1-deletion-and-n-extension ($n \geq 0$) cycles before the next spanning tree is obtained.

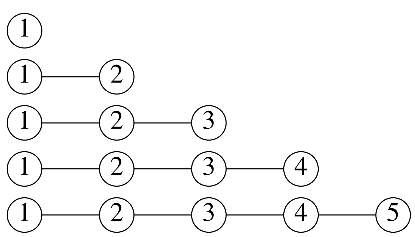


First computation path:	start → E → E → ···· → E → end	(generates a spaning tree starting from a single node)
Last computation path:	start → R → R → ···· → R → end	(reduces the last spanngin tree to a single node)

Example.

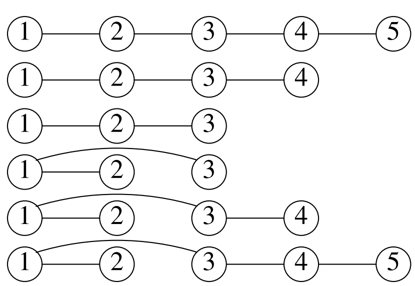
A connected graph G: 

start (no prevSavedTree):
E:
E:
E:
E:



end (return T): Generating the first spanning tree starting from one node.

start (with prevSavedTree):
R:
R:
E:
E:
E:



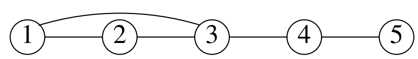
end (return T): Generating the second spanning tree by modifying the first; it uses three 1-deletion-and-n-extension cycles, with n = 0, 0, 3.

AN FSM WHICH AVOIDS EDGE-REPLACEMENTS THAT DO NOT LEAD TO SPANNING TREES

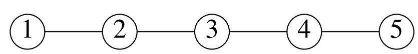
- An edge is replaced only if its deletion does not make the graph disconnected (in the presence of previous non-selected edges that are considerd "deleted", if any, in the current state)
- The most such recent edge in the current spanning tree is replaced first.
- The generation of each new spanning tree, following the first spanning tree, now involves a single n-deletion operation, where $n \geq 1$ edges are deleted from T (and those edges are put back to G for use in subsequent T's), followed by an equal number of extension operation.

Example. After generating the third spanning tree for the graph below, the alternative to the second edge (3,2) is (3,4). But since the current non-selected edges (1,2) makes (3,2) a cut-edge, the alternative (3,4) is not considered in the new FSM; otherwise, it would lead to a non-spanning tree with the edges {(1,3), (3,4), (4,5)}.

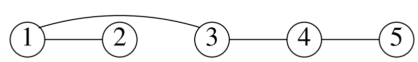
A connected graph G:



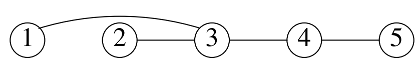
The first spanning tree:



The second spanning tree:

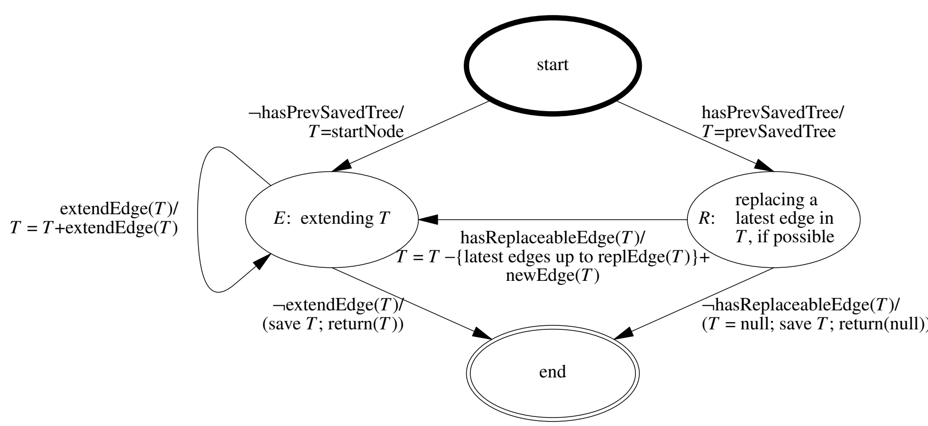


The third spanning tree:



The replacement of (2,3) by (1,3) then gives the second spanning tree.

The new FSM:

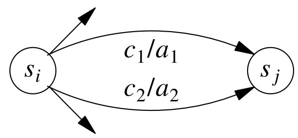


SIMPLIFYING AN FSM BY MERGING PARALLEL TRANSITIONS

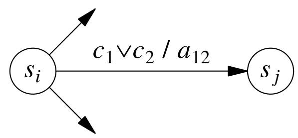
Parallel Transitions:

Two transitions t_1 and t_2 are *parallel* if they start at the same state s_i and they end in the same state s_j.

$t_1 = (s_i, c_1, a_1, s_j)$:
$t_2 = (s_i, c_2, a_2, s_j)$:



$t_{12} = (s_i, c_1 \vee c_2, a_{12}, s_j)$:



a_{12}: if (c_1) then a_1;
else a_2

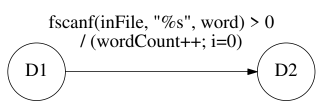
After merging. The action-part of a transition can be any computation and arbitrarily complex.

- If $a_1 = a_2 = a$, then the merged transition's condition/action is: $c_1 \vee c_2 / a$.

Def(t): For $t = (s_i, c_{ij}, a_{ij}, s_j)$, Def($t$) = Def($c_{ij}$) $\cup$ Def(a_{ij}), where Def(c_{ij}) is often empty.

Use(t): Use(c_{ij}) $\cup$ Use(a_{ij})

Example.



Def(t) = {inFile, word, wordCount, i}
Use(t) = {inFile, wordCount}

Def(t_{12}): Def(t_1) $\cup$ Def(t_2)

For example, if $x \in$ Def(a_1) − Def(a_2), and c_2 is true (hence c_1 is false), and Def(c_1) = $\varnothing$ = Def(c_2), then x is actually unchanged by t_{12} although $x \in$ Def(t_{12}). We can momentarily pretend that "$x = x$" is part of action a_2.

Use(t_{12}): Use(t_1) $\cup$ Use(t_2)

- A good FSM requires choosing conditions and actions that can be easily described (captured) in terms of well-understood concepts ("chunks" of information).

SIMPLIFYING AN FSM BY STATE-ELIMINATION

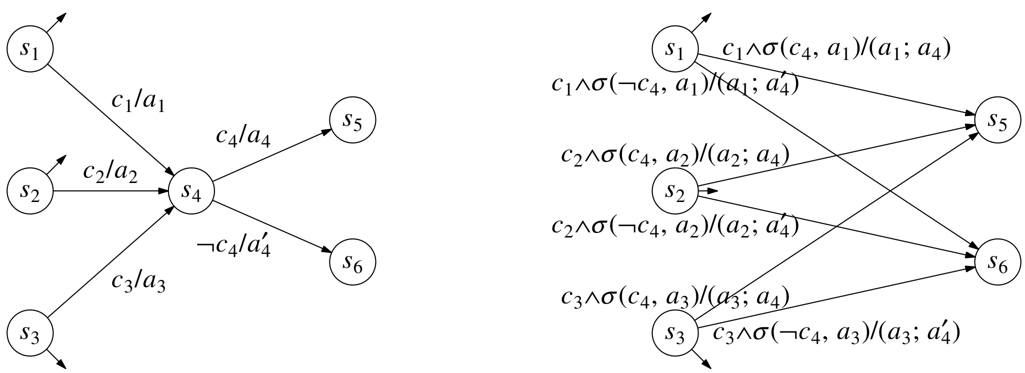
Elimination of s_j without a loop at it:

- $s_j \neq$ start-state or end-state.
- Need to eliminate each transition (s_i, s_j) to s_j.
- Elimination of (s_i, s_j) means the addition of transitions $\{(s_i, s_k)$: (s_j, s_k) is a transition from $s_j\}$, each with proper condition-action pair as shown below, where $a_{ij} = a(s_i, s_j)$ and $c_{jk} = c(s_j, s_k)$.

$$c(s_i, s_k) = c(s_i, s_j) \wedge \sigma(c_{jk}, a_{ij}) \quad \text{and} \quad a(s_i, s_k) = (a(s_i, s_j); a(s_j, s_k))$$

- The action a_{ij} modifies the condition c_{jk} to $\sigma(c_{jk}, a_{ij})$ by a substitution operation such that c_{jk} is expressed in terms of the values of variables *prior* to the action a_{ij}.

$$\begin{array}{rl} a(s_i, s_j): & (x = x + y;\ y = f(x, y, z)) \\ c_{jk}: & x > y \\ \hline \sigma(c_{jk}, a_{ij}): & (x + y) > f(x + y, y, z) \end{array}$$



(i) Before elimination of s_4. (ii) After elimination of s_4.

- Elimination of s_i is most useful if indeg$(s_i) = 1$ or outdeg$(s_i) = 1$.
- If a transition t_1 at s_i occurs only after the transition t_2 at s_i in each computation, then elimination of t_2 often produces useful simplification of FSM.

EXERCISE

1. Show the flowchart after replacing the for-loop in WordCharCount(·) function shown earlier by a suitable while-loop (eliminating the break-statement). Obtain the FSM for the new flowchart.
2. Show the FSM obtained by elimination of the state D1 in the FSM for WordCharCount-function. (What's the program corresponding to the simplified FSM?)
3. Is it possible to eliminate the transition (D3, D2) without eliminating the state D2? What would be the new FSM?

CHARACTERISTICS OF A POOR FINITE-STATE MODEL

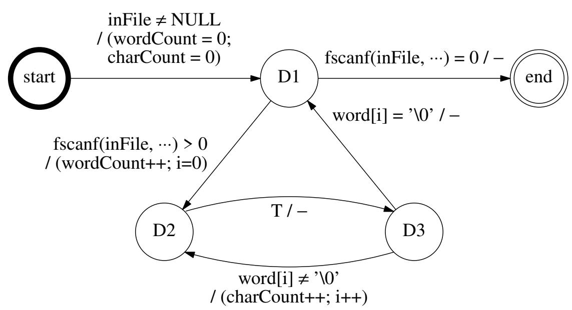
Characteristics:

1.1 Existence of an *unused* transition $\langle c(s_i, s_j), a(s_i, s_j)\rangle$, i.e., one which is never used for any valid input.

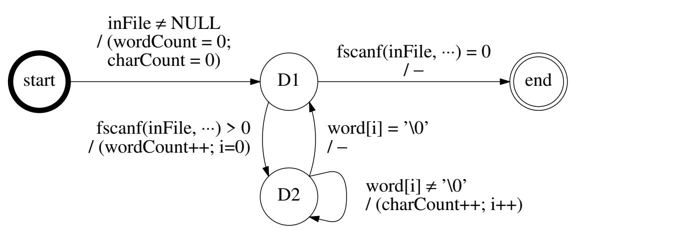
1.2 Existence of an *unused* state s_j, which is never reached for any valid input. (This means each transition to s_j is unused.) This is the same as saying s_j is logically unreachable although there is a path from start to end containing s_j.

2.1 Existence of *precedence* constraint among transitions from a state s_i: some transition from s_i cannot be used before some other transition(s) from s_i. (This arises from the existence of in feasible paths in the original program/algorithm.)

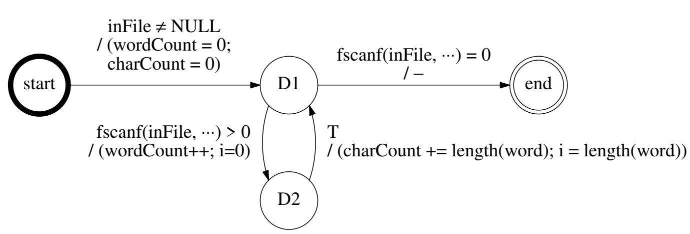
Example.



(i) After eliminating unused transition (D2, D1) and eliminating "$i \le$ WORDLEN" from the condition $c(D2, D3)$.



(ii) After elimination of the transition (D2, D3) and the state D3.

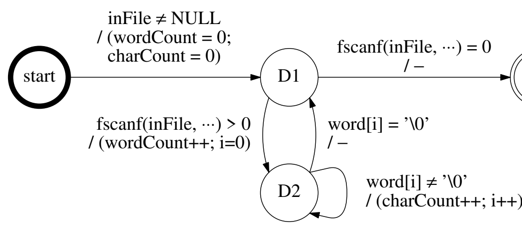


(iii) After elimination of (D2, D2) which precede (D2, D1). The elimination of (D2, D1) introduces a dependent transition (D2, end).

Question: If action $a(s_i, s_j)$ defines a variable x which is never used (in any condition or action reachable from s_j), then can we eliminate the definition of x in $a(s_i, a_j)$?

LOOP ELIMINATION

- This can be done provided we can determine a proper condition/action pair to represent loop-termination.

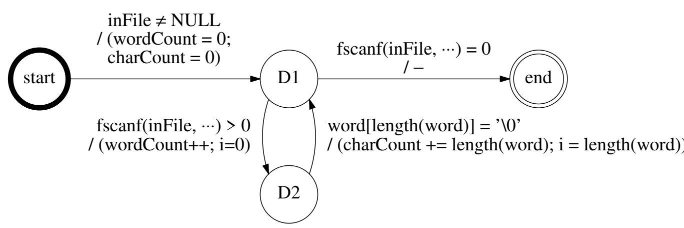


Code associated with loop at D2:

```
//note the initialization i = 0 in (D1,D2)
while (word[i] != '\0') {
      charCount++; i++;
}
```

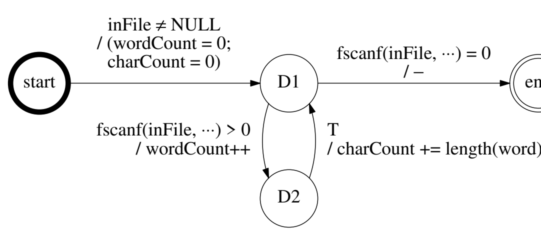
Assumptions: (word-array has size 1+WORDLEN) and (length(word) ≤ WORDLEN)

After loop-termination: charCount = charCount + length(word) and *i* = length(word)



An intermediate result of the elimination of loop (D2, D2).

- Note that *i* in "word[i] = '\0'" on the condition-part of the transition (D2, D1) has now been replaced with "length(word)" since the condition-part is verified before the action.
- We can actually replace the condition by true (T) since the while-loop shown above always terminates and the condition "word[length(word)] = '\0'" becomes true (T) on termination.
- We can drop the part "*i* = length(word)" from the action *a*(D2, D1) since this value of *i* is not used in any other condition or action. Likewise, we can drop the part "*i* = 0" from *a*(D1, D2).

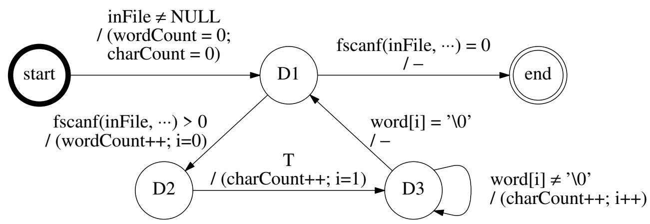


The final result of the elimination of loop (D2, D2).

Question: What would be the result of eliminating the transition (D1, D2) Can we now eliminate the resulting loop at D1? Does the elimination of (D2, D1) lead to a better alternative?

EXERCISE

1. Is the following a correct FSM for the WordCharCount-problem, i.e., does it represent a correct algorithm? If so, then identify as many problems as you can that make it a poor model; show the new FSM after eliminating those problems. (Assume that word-lengths are bounded by WORDLEN and the variable word has been allocated sufficient memory.)



2. Show the new FSM after eliminating the loop (D3, D3) and any subsequent simplifications of conditions and actions.

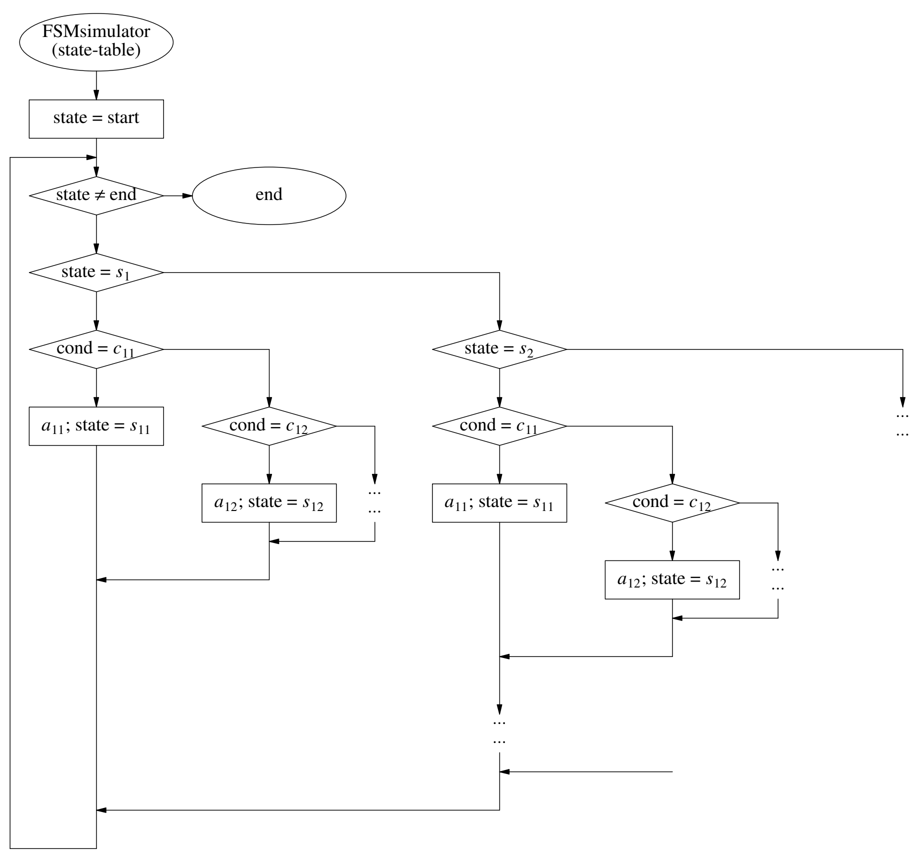
FSM-SIMULATOR

FSM-simulator:

Input: State-table and other inputs for the original source-program (including the function-parameters, if any).
Output: Same as that of the original source-program.

```
for (state = start; state != end; ) //state update is done after each action aij
    switch (state) {
           ...
           case si: switch (condition) {
                           ...
                           case cij: aij;
                                     state = sj;
                                     break;
                           ...
                    }
           ...
    }
```

Flowchart of FSM-simulator:



EXERCISE

1. Show a sketch of the FSM for the above flowchart, after a reasonable amount of simplification.

MODELING THE CRITICAL "NEXT-STOP" DECISION FOR A SIMPLE ELEVATOR PROBLEM

General Assumptions:

G1. The elevator-use of a person is controlled by his id-card which is validated for only one entry-exit(to destination)-entry(at destination)-exit sequence by a security officer at the ground floor.

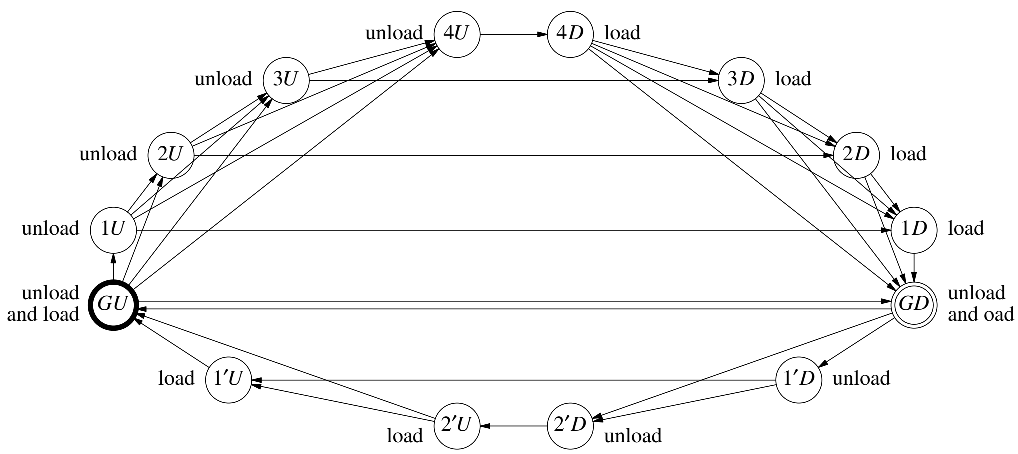
G2. A person registers the request to use the elevator on his current floor by inserting his id-card in a slot next to the elevator. (There are no floor buttons inside or outside the elevator. The floor-indicator light inside the elevator shows the floor number when the elevator stops.)

Assumptions about Elevator:

(A1) The elevator moves when an entry/exit-request (by persons outside/inside the elevator) is pending. It changes move-direction from UP (DN) only at a floor above (below) the ground-level.

(A2) The elevator does not "go through" the ground-floor without stopping there.

(A3) The load/unload operations are restricted as indicated in the FSM below since no one passes by a floor beyond his destination. The floor-indicator light on/off operations are not shown. (Floors below the ground level are indicated by 1′, 2′, etc.)



- The FSM shows the possible changes in (floorNum, moveDir) for a 7 (= 4+1+2) floor building, with 4 floors above the ground and 2 floors below. (It does not show the steps of the algorithm for changing these variables.)
- Since all transitions from a state s_i have the same operation, we associate it with s_i itself. Load/unload operations apply based on the pending entry/exit requests; they may depend on capacity constraints and floor-priorities.
- Since the condition for the transitions are generally complex, they are not shown in the state-diagram itself (see below for an example).

Question: (1) What would be wrong if we did not have the transition ($2U$, $2D$), keeping all other transitions as they are (even if we modify some of their codition/action pairs)?

(2) Are there precedent-constraints on the transitions?

Condition for the transition ($1U$, $3U$)**:**

(1) There is no exit-request for floor #2, and

(2a) There is either an exit-request for floor #3, or

(2b) There is no exit-request for floors #3 and #4, and there is no entry-request on floor #4, and there is an entry-request on floor #3. (In this case, no unloading operation is done at state $3U$.)

EXAMPLE OF LOAD/UNLOAD/RELOAD OPERATIONS

Left column: Persons outside the elevator on ground-floor with entry-requests, and persons at their destinations who have not put entry-reuests for their return journey.

Center column: Person in the elevator; persons with different exit-requests to their destinations are sorted in different lists.

Right column: Persons at their destiination who have entered their entry-requests for the return journey.

4			
3			
2			
1			
0	3(2), 2(4), 1(4)		
1′			
2′			

(i) Starting at GU.

	1, 2	
	3	

(ii) After load-operation at GU.

	1, 2	
3		

(iii) After unload-operation at $2U$.

1, 2		
3		
5(1′), 4(4)		

(iv) After unload-operation at $4U$; there are also two new arrivals.

1, 2		
3		
6(1′), 5(1′), 4(4)		

(v) Before load-operation at GD, after going via state $4D$; there is one new arrival.

1, 2		
3		
4(4)		
	5, 6	

(vi) After load-operation at GD.

2		1
		3
4(4)		
5, 6		

(vii) After unload-operation at $1'D$. Customers 1 and 3 are ready to return.

2	4	1
		3
5, 6		

(viii) After load-operation at GU after going via state $1'U$.

4, 2	1	
		3
7(2′)		
5, 6		

(ix) After unload-operation at $4U$ and load operations at $4D$; there is also a new arrival.

THE COMPLETE STATE FOR THE ELEVATOR-PROBLEM

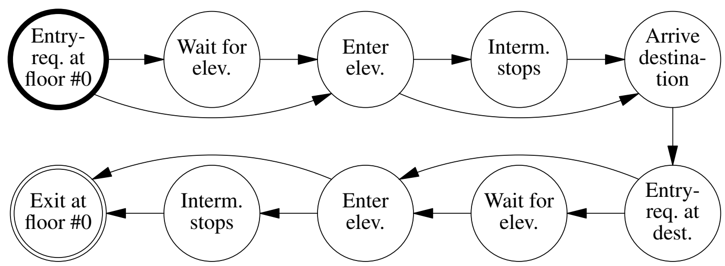
Components of Complete system-state:

(1) Elevator-state: its current floor# and move-direction.

(2) List of people (outside the elevator) with entry-requests and their destinations, one for each destination-floor. (Assume for simplicity that the departure time is determined after a person arrives at his destination.)

(3) List of people (inside the elevator) with exit-requests, one for each destination-floor.

FSM for a Person (approx.): The condition/action pairs are not shown.



Question: (1) What makes this an approximate description? How will you correct the problem?
(2) What is the condition for the two transitions at the start-state?
(3) Are there any precedence-constraints on the transitions here?

EXERCISE

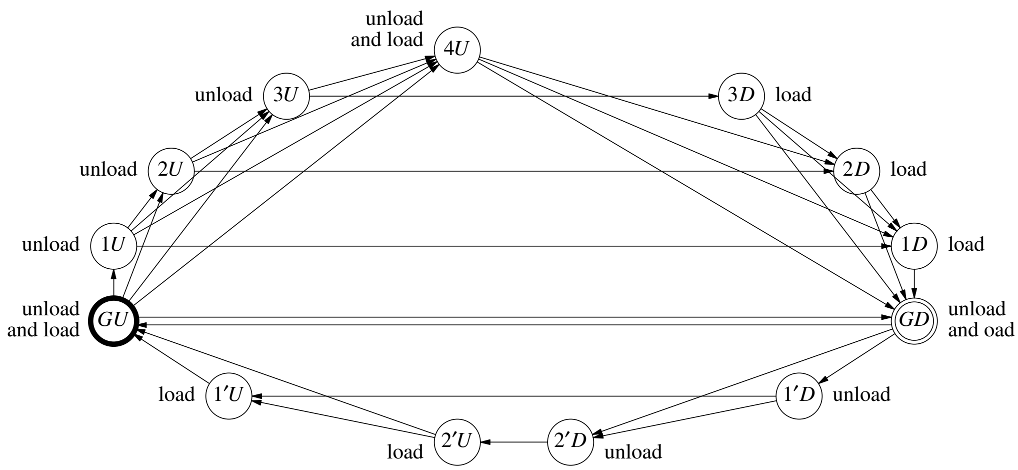
1. Which of the following conditions for various transitions at $3U$ and $3D$ are right? Show the corrected form, if it is not right. Note that if neither of $c(3U, 4U)$ or $c(3U, 3D)$ holds, then the elevators stays at $3U$. Similarly, the elevator stays at $3D$ if $\neg c(3D, 2D) \wedge \neg c(3D, 1D) \wedge \neg c(3D, GD)$ holds; however, it is not hard to see that if the elevator is at $3D$, then the above condition cannot be true and hence the elevator will not stay at $3D$.

$c(3U, 4U)$:	(1)	There is an exit-request for floor #4, or
	(2)	There is an entry-request on floor #4.
$c(3U, 3D)$:	(1)	$\neg c(3U, 4U)$, and
	(2)	There is an entry-request on one or more of the floors {#3, #2, #1, #0, #1′, #2′}. (The entry-request on floor #0 may be for going down or up.)
$c(3D, 2D)$:	(1)	There is an entry-request on floor #2.
$c(3D, 1D)$:	(1)	$\neg c(3D, 2D)$, and
	(2)	There is an entry-request on floor #1.
$c(3D, GD)$:	(1)	$\neg c(3D, 2D) \wedge \neg c(3D, 1D)$, and
	(2)	There is an entry-request on one or more of the floors {#0, #1′, #2′, #3, #4}.

2. What would go wrong if we delete the transition $(3U, 3D)$ and merge $3U$ and $3D$? (Hint: Look at the condition-action pairs associated with all transitions from $3U$ and all transitions from $3D$.)
3. Can we merge the states $4U$ and $4D$?
4. How would the FSM change if we give up the assumption (A1)? How about if we give up (A2)?
5. What are some advantages and disadvantages of (A3)? How will the FSM change if we give up the restriction that a person is not to pass a floor beyond his destination (see A3)?
6. Consider the function ElevatorNextState(systemState) based on the FSM shown earlier. Show a sketch of the algorithm and the FSM representation of it.

EXERCISE

1. Shown below is the states and transition (without the condition-parts) after eliminating the transition $(4U, 4D)$, using the standard method, and then eliminating the state $4D$. Note the new action associated with $4U$ Show the condition for the new transition $(4U, 3D)$.



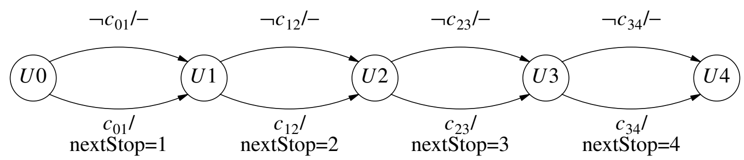
2. Can we carry this process and eliminate the states $3D$? Show the new transitions without the condition-parts.
3. What would the diagram look like if we eliminate also the states $2D$, $1D$, $1U$, $2U$, and $3U$ (and likewise the states $1'D$, $2'U$, and $1'U$)? Should we really do this − justify your yes/no answer. What is the moral of this exercise?

- State reduction at the cost of increasing complexity of condition-action pairs does not merit beyond a certain point.
- Clarity of the overall structure (symmetry, etc.) and the ease of understanding of the condition/actions is more important.
- This also helps modification and extension.

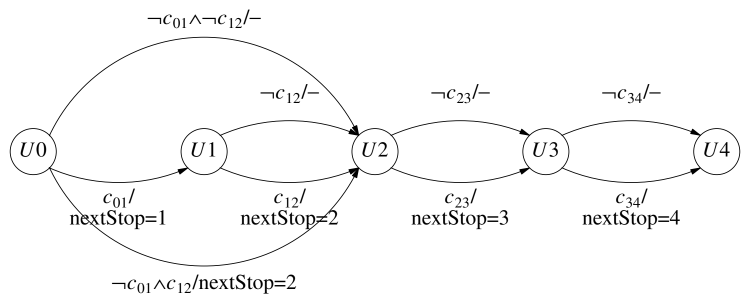
AN ALTERNATE NOTION OF STATE FOR THE ELEVATOR PROBLEM

State U_i: (1) Decide whether to stop at U_{i+1} (and unload) or not, for $i \leq 3$, on its upward move.
(2) The elevator may be at floor #i or at floor #i', $0 \leq i' < i$, on its upward move.

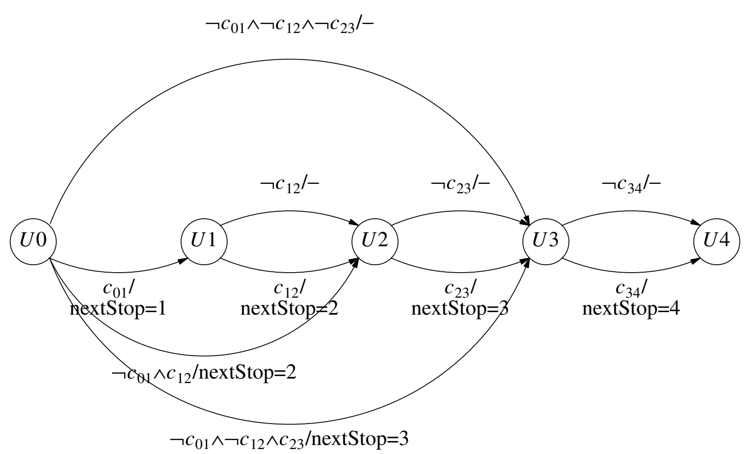
State U_4: (1) Decide whether to stop (and unload, if necessary) and load at floor #4 or not.
(2) The elevator may be at floor #i or at floor #i', $0 \leq i' < i$, on its upward move.



(i) Part of the FSM.



(ii) After elimination of transitions from ($U0$, $U1$) with no action.



(iii) After elimination of transition ($U0$, $U2$) with no-action.

In this way, we slowly transform the transitions at $U0$ and get back the transitions shown earlier. When the process is repeated for the states $U1$, $U2$, $\cdots$, the new transitions provide the alternate meaning of the states.

INVARIANCE PROPERTIES OF TRANSITION-ELIMINATION

Strong-component:

It is a maximal subset C of states such that for any two distinct states s_i, $s_j \in C$, there is an s_i-s_j path which is entirely contained in C.

- Two strong components have no states in common.
- The strong components form a partition of the states into disjoint subsets.

Def and Use: For a state s_i:

$$\mathrm{Def}(s_i) = \bigcup\{\mathrm{Def}(t_{ij}):\ t_{ij} \text{ is a transition at } s_i\}.$$
$$\mathrm{Use}(s_i) = \bigcup\{\mathrm{Use}(t_{ij}):\ t_{ij} \text{ is a transition at } s_i\}.$$

For a strong component C:

$$\mathrm{Def}(C) = \bigcup\{\mathrm{Def}(s_i):\ s_i \in C\}.$$
$$\mathrm{Usef}(C) = \bigcup\{\mathrm{Usef}(s_i):\ s_i \in C\}.$$

Invariance Properties:

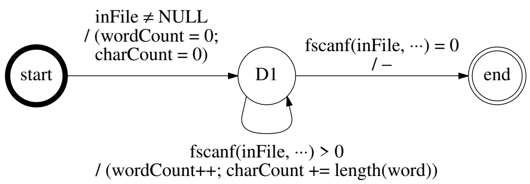
(1) If C is a strong component, then the elimination of a transition (s_i, s_j), s_i, $s_j \in C$ does not affect the strong component C, except that it may now equal $C - \{s_i\}$, if there are no other transition to s_j within C.

If $s_i \in C_i$, $s_j \in C_i$, and $C_i \neq C_j$, then the elimination of (s_i, s_j) does not change any of the strong components. (C_j becomes unreachable if $C_j = \{s_j\}$ and there were no other transitions to s_j.)

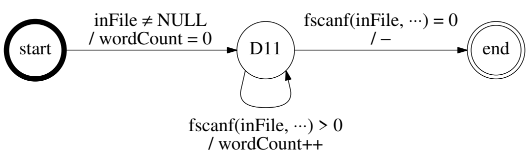
(2) No Def(C) and Use(C) are unaffected by the elimination of a transition.

COMPOSITION OF FSMs

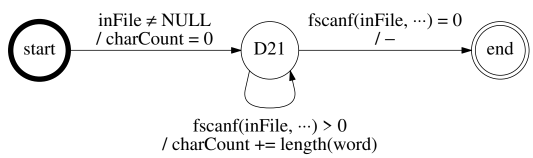
Consider the two simpler FSMs obtained from the final form of FSM for WordCharCount-function, by removing the actions related to charCount and wordCount, respectively, and then simpplifying the resulting FSMs.



(i) FSM for wordCount and charCount.



(iv) Simplified form of (ii).



(v) Simplified form of (iii).