Section 3: Model Checker

건국대학교 컴퓨터공학부
2020147178 김건하

A brief view

- We wish to determine whether formula f_0 is true in finite structure M.
- How?
- The 1st stage processes all sub-formulas of length 1
    - The 2nd stage processes all sub-formulas of length 2
    - …..
    - At the end of the ith stage, each state will be labeled with the set of all sub-formulas of length less than or equal to i that are true in the state.
    - $label(s)$: this set for state s
- When the algorithm terminates at the end of the stage $n = length(f_0)$
    - For all state s,M, $s \vDash f \ iff \ f \in \ label(s)$ for all sub-formulas f of f_0

Some Primitives

- $arg1(f)$ and $arg2(f)$ give the 1st and 2nd arguments of a two-argument temporal operator; thus, if f is $A[f_1\ U f_2]$, then $arg1(f) = f_1$, $arg2(f) = f_2$
- labeled (s, f) will return true (false) if state s is (is not) labeled with formula f
- add_label (s, f) adds formula f to the current label of state s

State Labeling Algorithm

- We have to handle seven cases
  - $\neg f1,\ f1 \wedge f2,\ AXf_1,\ EXf_1,\ A[f_1\, U f_2]\ or\ E[f_1\, U f2]$
- Consider $f = A[f_1\, U f_2]$
- We're gonna use Depth-First-Search
- marked[s] : check if state s have been visited
- ST : auxiliary Stack variable introduced for proof of correctness of the algorithm
- stacked(s) : check if state s is currently on the stack ST

- **procedure** label_graph(f)
  begin
    …..
    [main operator is AU]
    begin
      ST := empty_stack;
      for all $s \in S$ do marked(s) := false;
      L: **for all** $s \in S$ do
      if ¬marked(s) **then** $au(f, s, b)$
    end
      …..
  end

Procedure $au(f, s, b)$

- Recursive procedure
- Search for formula f starting from state s.
- When au terminates, b will be set to true iff $s \models f$

Procedure $au(f, s, b)$

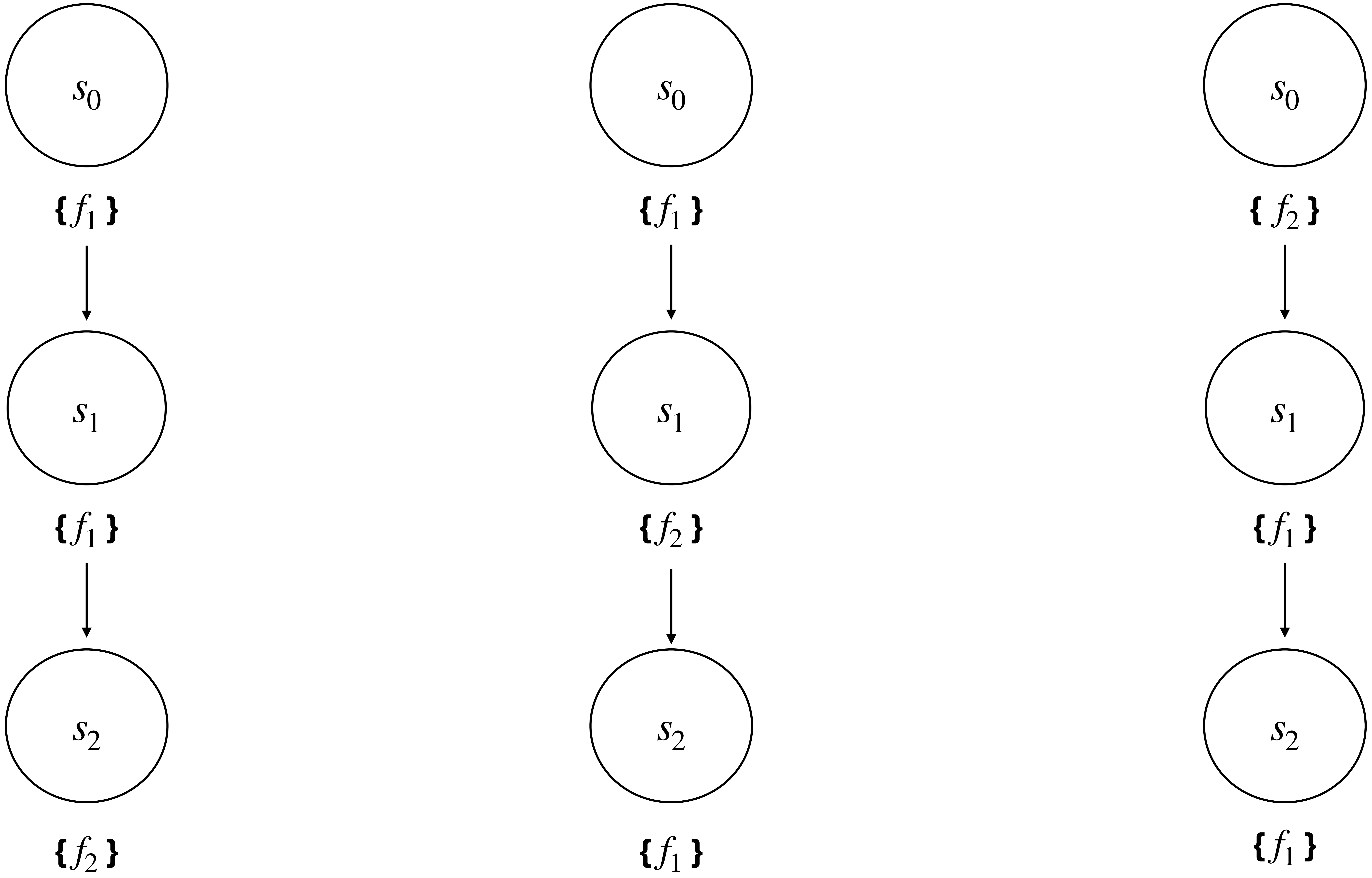
- Assume that s is marked
- If s is on the stack, there is a cycle which is arg1(f) holds but f is never fulfilled
- Suppose there exists a path $(s_1, s_2, \cdots, s_m, s_k)$ in the state graph such that $1 \leq k \leq m$ and $\forall i[1 \leq i \leq m \rightarrow s_i \vDash \neg f_2]$, then $s_k \vDash \neg A[f_1\, U f_2]$
- Or, we have already completed a DFS from s, and f is false at s
- Both of this two cases set b false

- **procedure** $au(f, s, b)$
  begin
  if marked(s) **then**
      begin
          if labeled(s, f) **then**
              begin b:= true; **return end**;
          b:= false; **return**
      end;

Procedure $au(f, s, b)$

- Mark state s
- If f_2 is true at s, f is true at s
- If f_1 is not true at s, then f is not true at s
- Why??

- $s_0 \vDash A[f_1\, U f_2]$ *iff for all paths* $(s_0, s_1, \cdots)$, $\exists i[i \geq 0 \wedge s_i \vDash f_2 \wedge \forall j[0 \leq j < i \rightarrow s_j \vDash f_1]]$



- …

  marked(s) := true;

  if labeled (s, arg2(f)) **then**

  begin add_label (s, f); b:= true; **return end**;

  else if ¬labeled (s, arg1(f)) **then**

  begin b:= false; **return end**;

Procedure $au(f, s, b)$

- Now we know that f_1 is true at s and that f_2 is not
  $->$ $P(s) = \{f_1\}$ ($P : S \to 2^{AP}$)

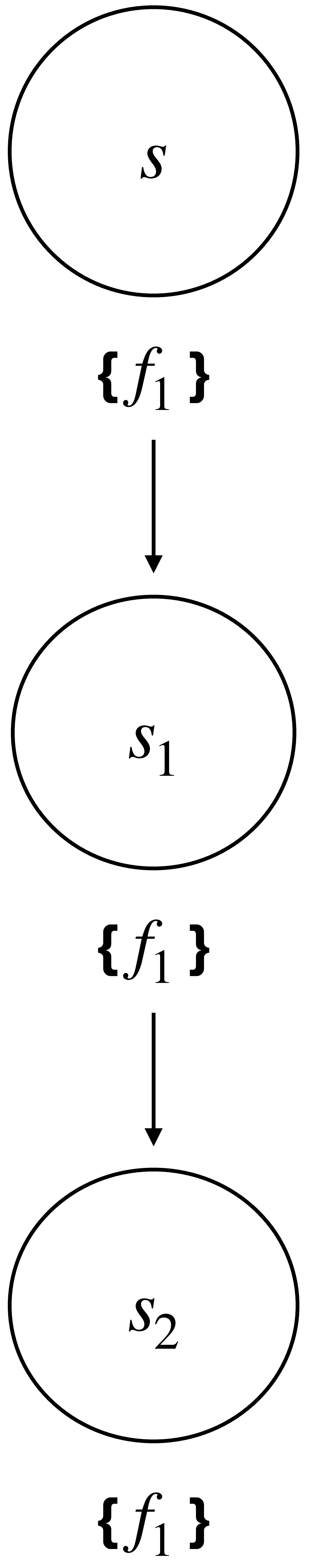
- Check to see if f is true at all successor state of s.

- Why?

- Remind the definition of formula $f = A[f_1 \, U f_2]$

- $s_0 \vDash A[f_1\, U f_2]$ *iff for all paths* $(s_0, s_1, \cdots)$, $\exists i[i \geq 0 \wedge s_i \vDash f_2 \wedge \forall j[0 \leq j < i \rightarrow s_j \vDash f_1]]$

- Check that $P(s) = \{f_1\}$



- …

```
push(s, ST);
for all s1 ∈ successors(s) do
  begin
    au(f, s1, b1);
    if ¬b1 then
      begin pop(ST); b := false; return end;
  end;
pop(ST); add_label(s, f); b:= true; return;
end of procedure au;
```

Time Complexity

- Assume that the states of the graph are already correctly labeled with f_1 and f_2 by this algorithm
- Time Complexity: $O(|S| + |R|)$
- Some people may be more familiar with $O(V + E)$

CTL formulas: arbitrary nesting of sub-formulas

- If we write formula f in prefix notation and count repetitions, the number of sub-formulas of f = length(f)

- Length (f) = the number of operands + the number of operators

- What about Path quantifier?

- Consider $f = (AU(NOT\ X)(OR\ Y\ Z))$

- Length (f) = 6

Numbering of sub-formulas

- Assume that formula f is assigned the integer i

- If f is unary like op f_1, we assign the integers $[i+1,\ i+length(f_1)]$ to the sub-formulas of f_1

- If f is binary like op $f_1 f_2$, we assign the integers $[i+1,\ i+length(f_1)]$ to the sub-formulas of f_1,
$(i+length(f_1),\ i+length(f_1)+length(f_2)]$ to the sub-formulas of f_2

Two arrays: $nf\,[\ ]$ and $sf\,[\ ]$

- $nf\,[1 : length(f)]$
- $nf\,[i]$ == ith sub-formula of f
- $sf\,[1 : length(f)]$
- $sf\,[i]$ == list of the numbers assigned to the immediate sub-formulas of $nf\,[i]$

Example

- Consider $f = (AU(NOT\,X)(OR\,Y\,Z))$
- In infix notation, $f = (A(NOT\,X)U(Y\,OR\,Z))$
- Assign 1 to f
- Let $f_1 = (NOT\,X), f_2 = (Y\,OR\,Z)$
- Assign 2 to $f_1 = NOT\,X$
- f_1 can be divided into NOT and X
- Assign 3 to X
- Do the same to the f_2 as f_1

Example

- So, the two arrays are

$$nf[1] = (AU(NOT\,X)(OR\,Y\,Z)) \qquad sf[1] = (2,4)$$
$$nf[2] = (NOT\,X) \qquad sf[2] = (3)$$
$$nf[3] = X \qquad sf[3] = nil$$
$$nf[4] = (OR\,Y\,Z) \qquad sf[4] = (5,6)$$
$$nf[5] = Y \qquad sf[5] = nil$$
$$nf[6] = Z \qquad sf[6] = nil$$

Question

- Does operand(in this example, X, Y and Z) should be Atomic Proposition or it can be CTL formula?
- If operand can be CTL formula, how can we distinguish sub-formula from operand?

Using Two arrays: $nf[\ ]$ and $sf[\ ]$

- Given the number of a formula f we can determine in constant time the operator of f and the numbers assigned to its arguments.

- How?

- Determine the operator of f : lookup $nf[i]$ with $sf[i]$

- Determine the numbers assigned to its arguments : lookup $sf[i]$

- If $sf[i]$ is nil, then i is the number assigned to its arguments

Implementing procedures

- We can implement the procedures "labeled" and "add_label" by using a bit array $L[s][length(f)]$

- add_label(s, fi) sets $L[s][fi]$ true

- labeled(s, fi) returns $L[s][fi]$

Conclusion

- Now, we can handle an arbitrary CTL formula f with numbering of sub-formulas of f
- for fi := length(f) step -1 until 1 do
  label_graph(fi);
- Time Complexity: $O(length(f) \times (|S| + |R|))$

Example

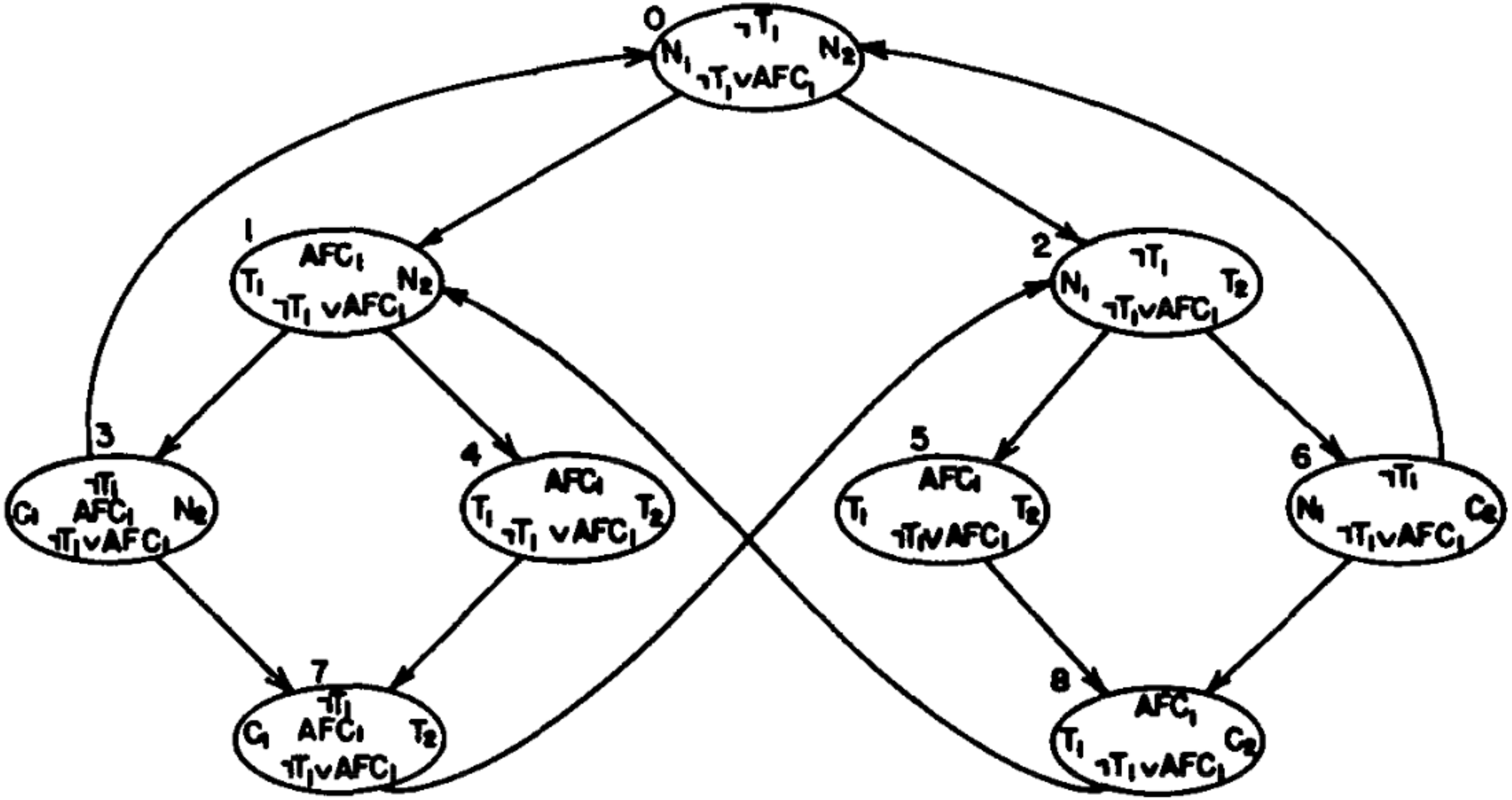


Fig. 4. Global state transition graph after termination of the model checking algorithm.

Example

- To establish absence of starvation for process 1, we consider the CTL formula $T_1 \rightarrow AFC_1$ or $\neg T_1 \vee AFC_1$
- Consider $f = \neg T_1 \vee AFC_1$
- Sub-formulas of f: $\neg T_1 \vee AFC_1,\ \neg T_1,\ T_1,\ AFC_1,\ C_1$
- On termination, every state will be labeled with $\neg T_1 \vee AFC_1$
- That is, $s_0 \vDash AG(T_1 \rightarrow AFC_1)$