Introduction to Formal Methods

Chapter 3. Model Checking

Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr

3. Model Checking

- Motivation:
  - Describe the principles underlying the algorithms used for model checking

  - The algorithm
    - Can find out whether a given automaton satisfies a given temporal formula
    - Different algorithms for CTL and PLTL

- Organization of Chapter 3
  - Model Checking CTL
  - Model Checking PLTL
  - The State Explosion Problem

3.1 Model Checking CTL

- Model checking algorithm for CTL
  - Developed in 1980s
  - Runs in time linear in each of its components (automaton and CTL formula)
  - Relies on the fact that CTL can only express state formulas

- Basic principles
  - procedure **marking**
    - Starting from a CTL formula ϕ
    - Mark for each state q of the automaton and for each sub-formula ψ of ϕ,
    - Whether ψ is satisfied in state q

- Correctness of the algorithm
  - …
  - Hence, the marking of q is correct.

- Complexity of the algorithm
  - Model checking " does $A, q_0 \models \Phi$? " for a CTL formula ϕ
  - can be solved in time O($|A| \times |\phi|$)
    - O($|A|$) : for marking the automaton
    - O($|\phi|$) : for each sub-formula in ϕ
  - Linear!!!

```
procedure marking(phi)

  case 1: phi = P
    for all q in Q, if P in l(q) then do q.phi := true,
                                  else do q.phi := false.

  case 2: phi = not psi
    do marking(psi);
    for all q in Q, do q.phi := not(q.psi).

  case 3: phi = psi1 /\ psi2
    do marking(psi1); marking(psi2);
    for all q in Q, do q.phi := and(q.psi1, q.psi2).

  case 4: phi = EX psi
    do marking(psi);
    for all q in Q, do q.phi := false;        /* initialisation */
    for all (q,q') in T, if q'.psi = true then do q.phi := true.

  case 5: phi = E psi1 U psi2
    do marking(psi1); marking(psi2);
    for all q in Q,
      q.phi := false; q.seenbefore := false;/* initialisation */
    L := {};                      /* L: states to be processed */
    for all q in Q, if q.psi2 = true then do L := L + { q };
    while L nonempty {
      draw q from L;                              /* must mark q */
      L := L - { q };
      q.phi := true;
      for all (q',q) in T {         /* q' is a predecessor of q */
        if q'.seenbefore = false then do {
          q'.seenbefore := true;
          if q'.psi1 = true then do L := L + { q' };
        }
      }
    }
```

```
case 6: phi = A psi1 U psi2
  do marking(psi1); marking(psi2);
  L := {}                             /* L: states to be processed */
  for all q in Q,
    q.nb := degree(q); q.phi := false;         /* initialisation */
  for all q in Q, if q.psi2 = true then do L := L + { q };
  while L nonempty {
    draw q from L;                                   /* must mark q */
    L := L - { q };
    q.phi := true;
    for all (q',q) in T {             /* q' is a predecessor of q */
      q'.nb := q'.nb - 1;                              /* decrement */
      if (q'.nb = 0) and (q'.psi1 = true) and (q'.phi = false)
        then do L := L + { q' };
    }
  }
```

3.2 Model Checking PLTL

- Model checking algorithm for PLTL
  - Developed in 1980s, but too technical to cover in this course

  - PLTL uses path formulas
  - No longer possible to rely on marking the automaton states

  - A finite automaton will generally give rise to infinitely many different executions, themselves often infinite in length

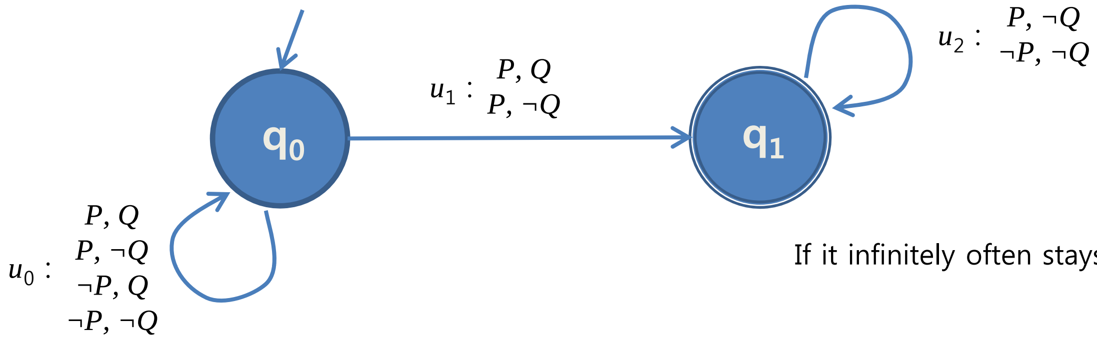
  - Hence, PLTL uses a <u>language theory</u> : ω-regular expression
    - An extension of a regular expression
    - "*" : an arbitrary but finite number of repetitions
      - (a b* + c)*
    - "ω": an infinite number of repetitions

- Basic principle
  - Model checking " does $A \models \phi$? " for a PLTL formula ϕ
  - Reduces to a " Are all the execution of A of the form described by ε_{ϕ} ? "

  - A PLTL model checker construct an automaton $B_{\neg\phi}$ (recognizing executions which do not satisfy ϕ)
  - Strongly synchronize A and $B_{\neg\phi}$ → $A \otimes B_{\neg\phi}$

  - Finally reduces to " Is the language recognized by $A \otimes B_{\neg\phi}$ empty ?"

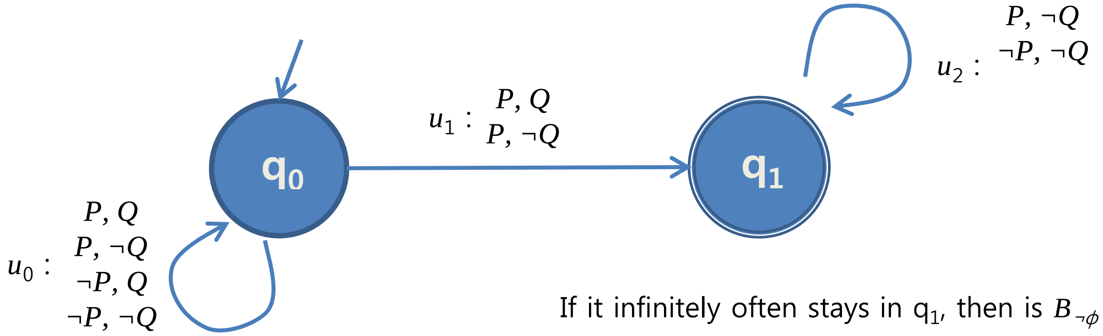
- A simple example
  - ϕ : G($P \Rightarrow$ XF Q) → any occurrence of P must be followed (later) by an occurrence of Q
  - $B_{\neg\phi}$ → there exists an occurrence of P after which we will never again encounter Q



If it infinitely often stays in q_1, then is $B_{\neg\phi}$ satisfied.

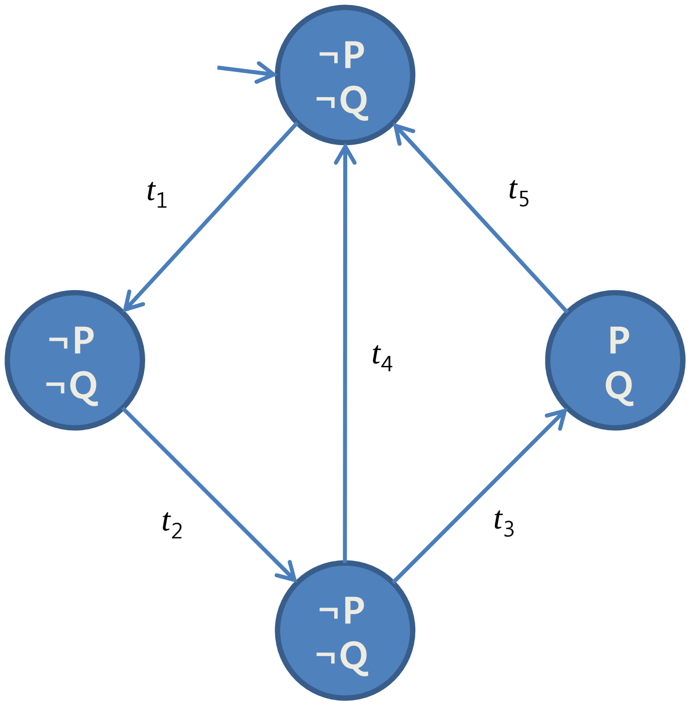
$\phi : \mathrm{G}(P \Rightarrow \mathrm{XF}\ Q)$

$B_{\neg\phi}$:



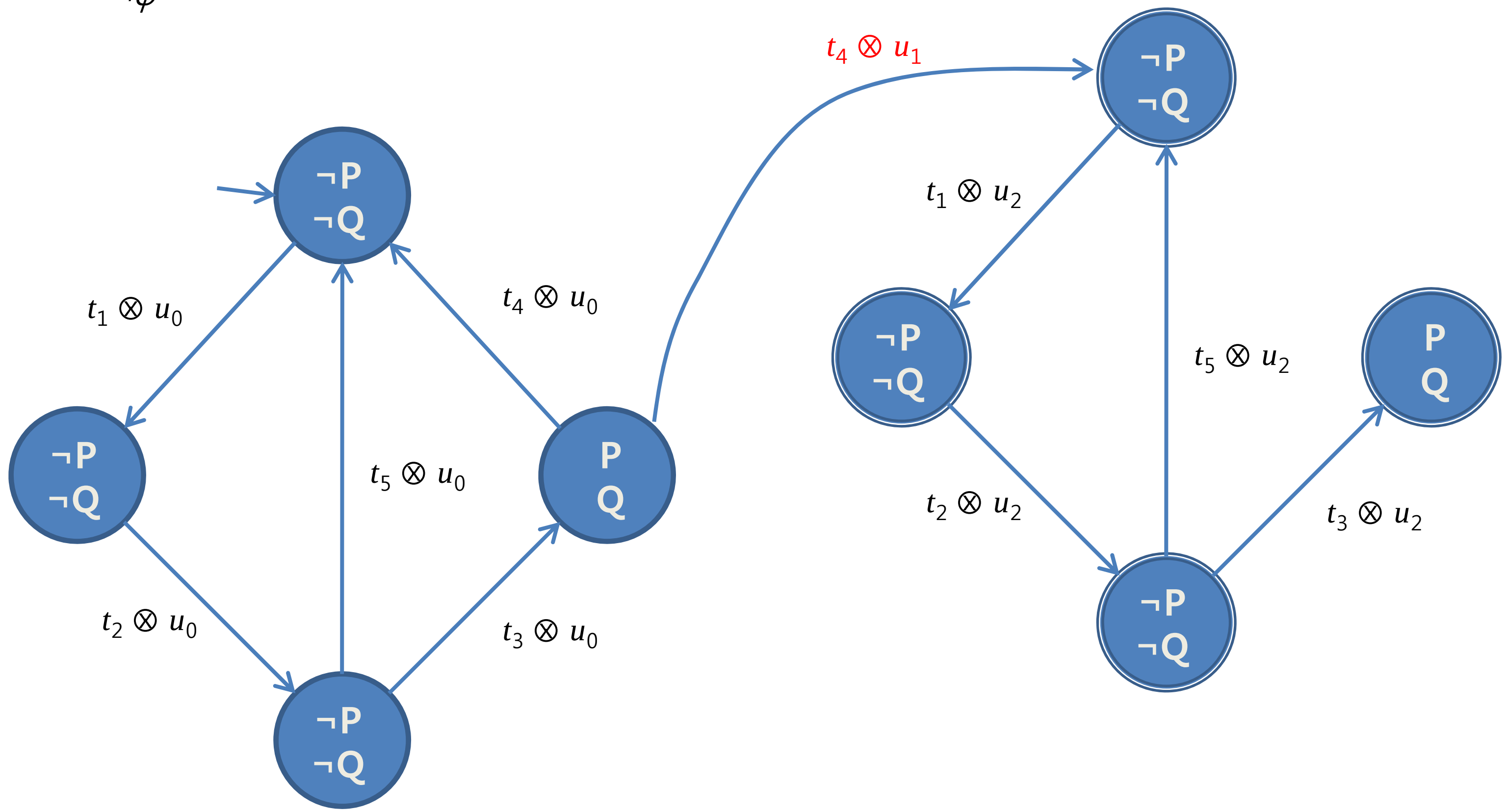
If it infinitely often stays in q_1, then is $B_{\neg\phi}$ satisfied.

A :



" does $A \nvDash \phi$? "

$A \otimes B_{\neg\phi}$:



There are behaviors of A accepted by $A \otimes B_{\neg\phi}$

→ The language recognized by $A \otimes B_{\neg\phi}$ is nonempty

→ $A \nvDash \phi$

- Construction of $B_{\neg\phi}$
  - Very difficult technically
  - Automaton $B_{\neg\phi}$ must in general be able to recognize infinite words
    → Büchi automata

- Complexity of the algorithm
  - $B_{\neg\phi}$ has size $O(2^{|\phi|})$ in the worst case
  - $A \otimes B_{\neg\phi}$ has size $O(|A| \times |B_{\neg\phi}|)$
  - If $A \otimes B_{\neg\phi}$ fits in computer memory, we can determine it in time $O(|A| \times |B_{\neg\phi}|)$

  - Model checking "does $A, q_0 \models \phi$?" for a PLTL formula ϕ can be done in time $O(|A| \times 2^{|\phi|})$

- Reachability analysis
  - We can say that $B_{\neg\phi}$ observes the behavior of A when the two automata are synchronized.
  - Observable automata = formal specification of the desired property
    - UPPAAL
    - SPIN

3.3 The State Explosion Problem

- State explosion problem
  - The main obstacle encountered by model checking algorithms
  - Indeed, the algorithms rely on explicit construction of the automaton A
    - Traversal and marking (in case of CTL)
    - Synchronization with $B_{\neg\phi}$ and seeking of reachable states and loops (in case of PLTL)
  - In practice, the number of states of A is quickly very large
  - If we use values that are not priori bounded (integers, a waiting queue, etc.), we cannot even apply it
  - Explicit model checking → Symbolic model checking (Chapter 4)