# T-79.186 Reactive Systems

## *Automata on Infinite words*

***Spring 2005, Lecture 6***

Keijo Heljanko

`Keijo.Heljanko@tkk.fi`

## Automata on Infinite Words

- To handle model checking of linear time temporal logic $LTL$ it is natural to use automata. However, these automata will accept infinite words (strings/sequences) instead of finite words.
- These automata are very closely related to finite state automata (on finite words). Note, however, that several of the used definitions for them are subtly different than for finite state automata.
- The most widely used class of automata on infinite strings is called Büchi automata.

## Büchi Automata

The definition of Büchi automata is identical to the definition of a finite state automata.

**Definition 1** *A (nondeterministic) Büchi automaton* $\mathcal{A}$ is a tuple $(\Sigma, S, S^0, \Delta, F)$, where

- $\Sigma$ is a finite *alphabet*,
- $S$ is a finite set of *states*,
- $S^0 \subseteq S$ is set of *initial states*,
- $\Delta \subseteq S \times \Sigma \times S$ is the *transition relation*, and
- $F \subseteq S$ is the set of *accepting states*.

The meaning of the transition relation $\Delta \subseteq S \times \Sigma \times S$ is the following: $(s, a, s') \in \Delta$ means that there is a move from state $s$ to state $s'$ with symbol $a$.

The definition of the language accepted by the automaton differs from FSAs.

# Language of Büchi Automaton

A Büchi automaton $\mathcal{A}$ accepts a set of infinite words $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ called the *language* accepted by $\mathcal{A}$, defined as follows:

- A *run* $r$ of $\mathcal{A}$ on an infinite word $a_0, a_1, \ldots \in \Sigma^\omega$ is an infinite sequence $s_0, s_1, \ldots$ of states in $S$, such that $s_0 \in S^0$, and $(s_i, a_i, s_{i+1}) \in \Delta$ for all $i \geq 0$.
- Let $inf(r)$ denote the set of states appearing infinitely often in the run $r$. The run $r$ is *accepting* iff $inf(r) \cap F \neq \emptyset$. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ iff $\mathcal{A}$ has an accepting run on $w$.

The language of $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^{\omega}$ is the set of infinite words accepted by the Büchi automaton $\mathcal{A}$.

A language of automaton $\mathcal{A}$ is said to be *empty* when $\mathcal{L}(\mathcal{A}) = \emptyset$.

## Checking Emptiness

- It is easy to check whether $\mathcal{L}(\mathcal{A}) \neq \emptyset$ by using the following observation: The language of the Büchi automaton is non-empty iff from some initial state $s \in S^0$ an accepting state $s'$ can be reached, such that $s'$ can reach itself by a non-empty sequence of transitions.
  (I.e., there should be a path from $s'$ back to itself in $\Delta$ which contains at least one edge.)
- The check above can be easily made by a linear time algorithm. (We'll come back to that later.)

# Operations for Büchi Automata

- We will now start defining the Boolean operators on Büchi automata:
  $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ and $\mathcal{A} = \mathcal{A}_1 \cap \mathcal{A}_2$.
- We will not be able to show $\mathcal{A} = \overline{\mathcal{A}'}$ in this course due to its technical complexity, but it can also be done. Thus also Büchi automata are closed under the Boolean operations.

The $\cup$ for Büchi automata is identical with the FSAs.

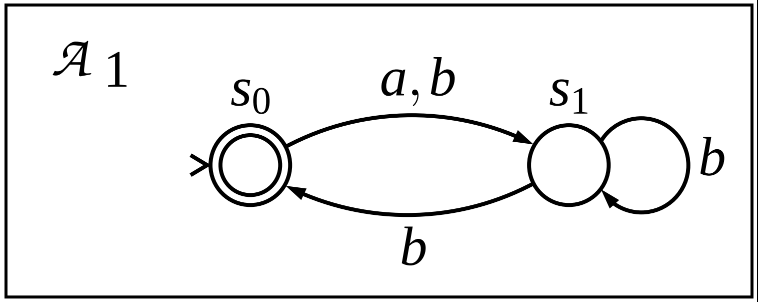
**Definition 2** Let $\mathcal{A}_1 = (\Sigma, S_1, S_1^0, \Delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, S_2, S_2^0, \Delta_2, F_2)$ be two Büchi automata.
We define the *union* Büchi automaton to be $\mathcal{A} = (\Sigma, S, S^0, \Delta, F)$, where:

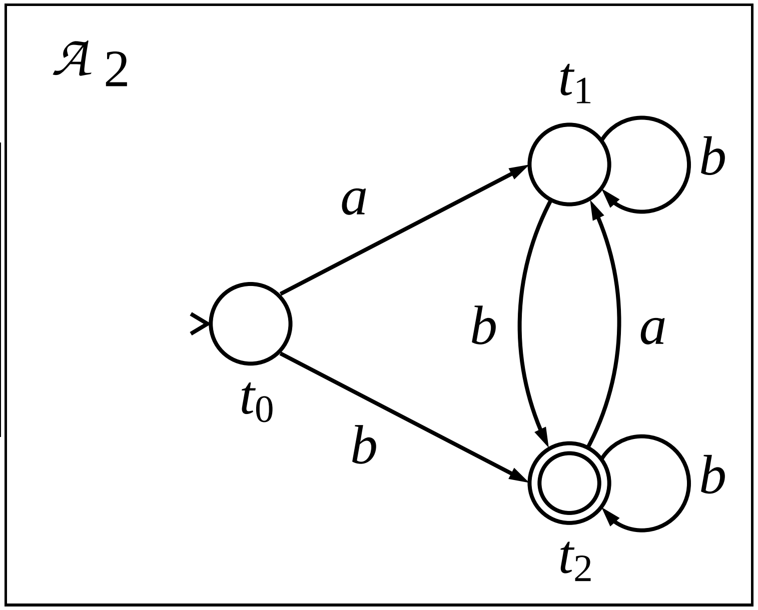
- $S = S_1 \cup S_2$,
- $S^0 = S_1^0 \cup S_2^0$,
- $\Delta = \Delta_1 \cup \Delta_2$, and
- $F = F_1 \cup F_2$.

Now for the union Büchi automaton $\mathcal{A}$ (also denoted by $\mathcal{A}_1 \cup \mathcal{A}_2$) it holds that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

# Example: Operations on BAs

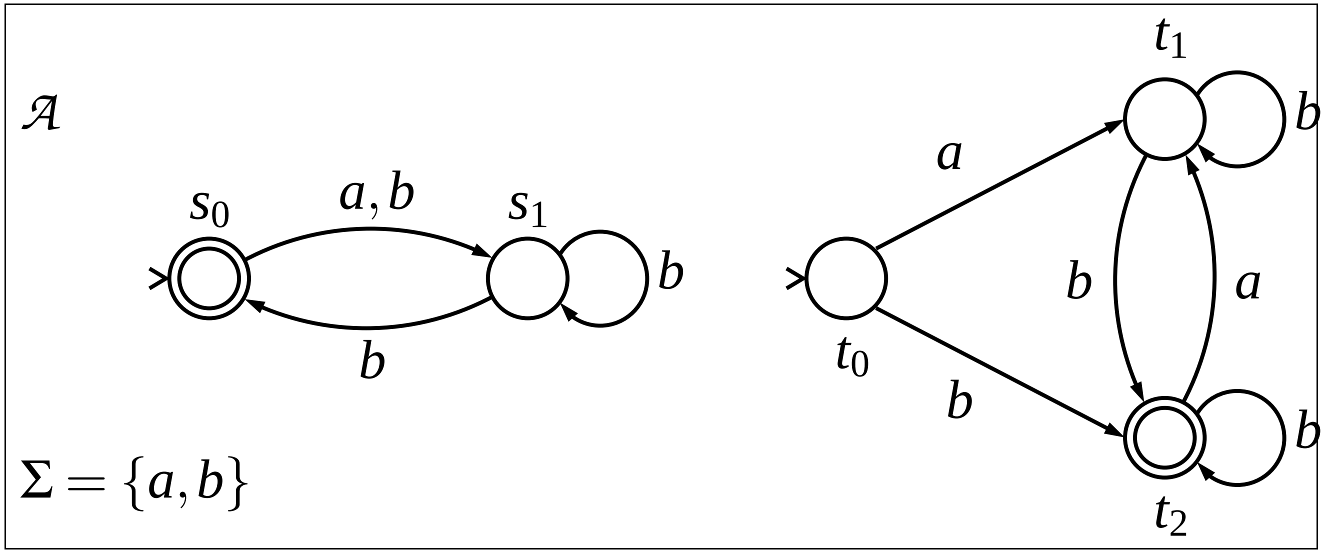
Consider the following Büchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$, both over the alphabet $\Sigma = \{a, b\}$.





## Example: Union of Büchi Automata

The following Büchi automaton $\mathcal{A}$ is their union, in other words $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$.



## Intersection of Büchi Automata

The $\cap$ definition for Büchi automaton *differs* from the FSA version!

If you use either FSA or Büchi definition in the wrong context, you will get *incorrect results*!

## Product of Büchi Automata

**Definition 3** Let $\mathcal{A}_1 = (\Sigma, S_1, S_1^0, \Delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, S_2, S_2^0, \Delta_2, F_2)$ be Büchi automata. We define the *product* Büchi automaton to be $\mathcal{A} = (\Sigma, S, S^0, \Delta, F)$, where:

- $S = S_1 \times S_2 \times \{1, 2\}$,
- $S^0 = S_1^0 \times S_2^0 \times \{1\}$,
- $F = F_1 \times S_2 \times \{1\}$, and

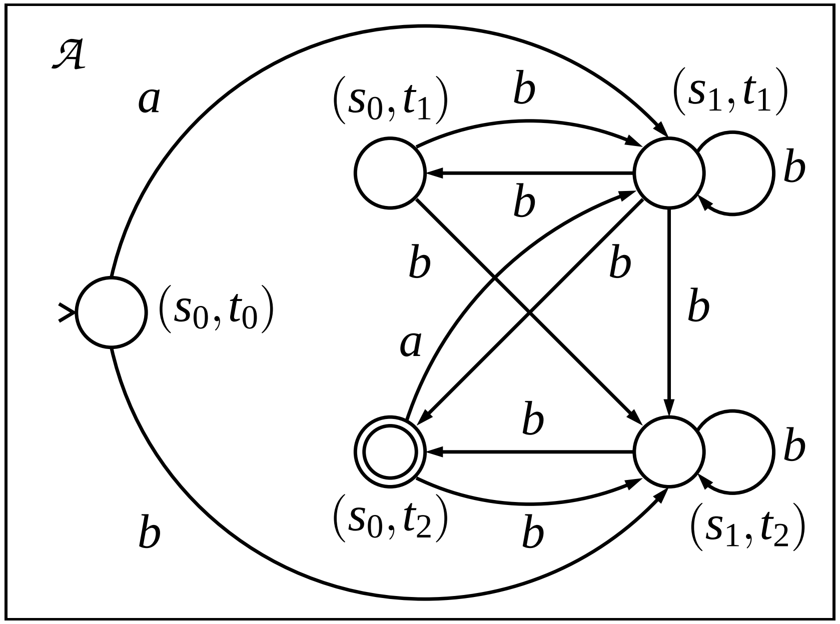
## Product of Büchi Automata cont.

- for all $s, s' \in S_1, t, t' \in S_2, a \in \Sigma, i, j \in \{1, 2\}$: $((s,t,i), a, (s',t',j)) \in \Delta$ iff $(s,a,s') \in \Delta_1$, $(t,a,t') \in \Delta_2$, and:
  a) ($i = 1$, $s \in F_1$, and $j = 2$), or
  b) ($i = 2$, $t \in F_2$, and $j = 1$), or
  c) (neither a) or b) above applies and $j = i$).

Now for the product Büchi automaton $\mathcal{A}$ (also denoted by $\mathcal{A}_1 \cap \mathcal{A}_2$ or $\mathcal{A}_1 \times \mathcal{A}_2$) it holds that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

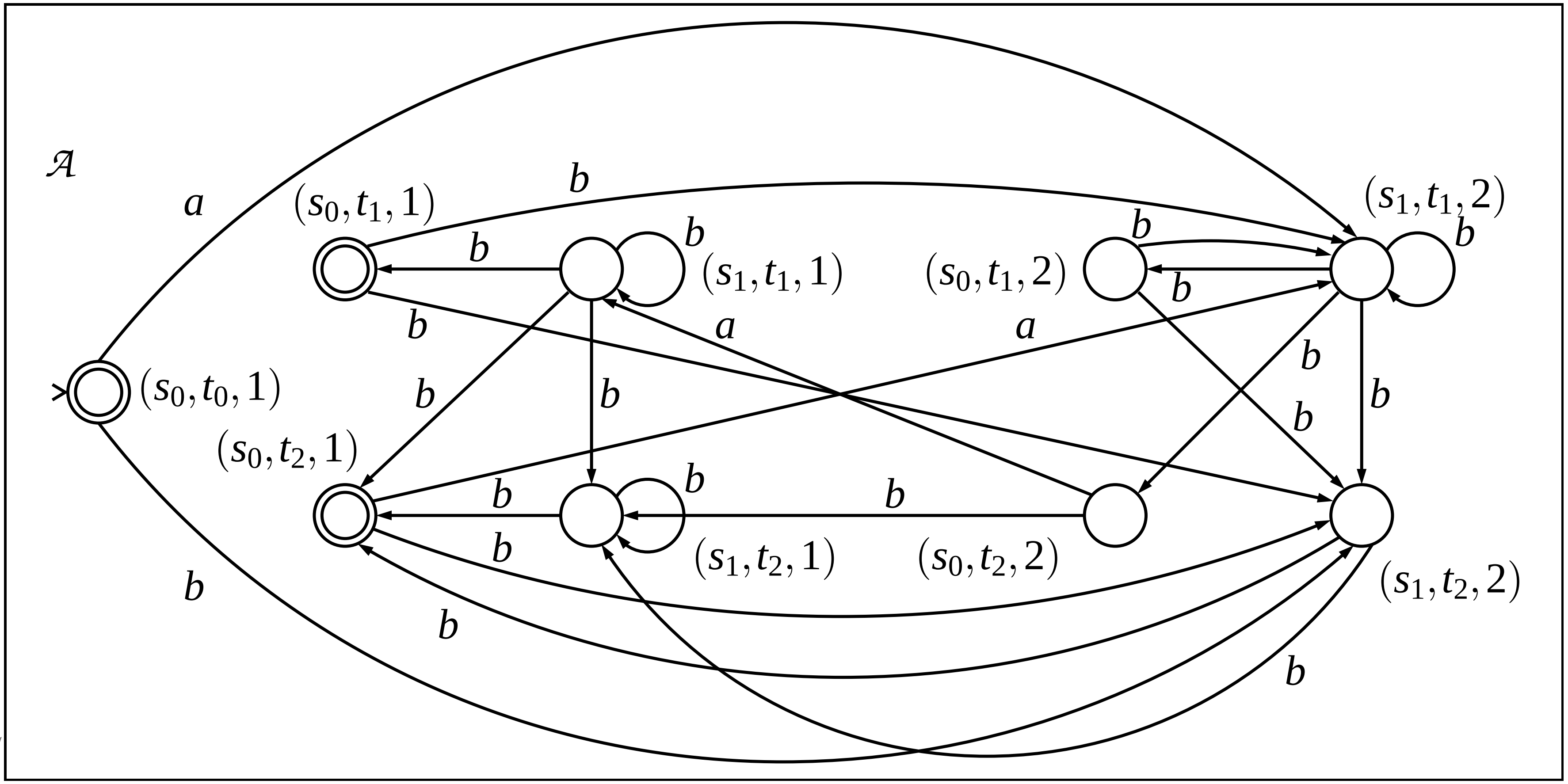
# Example: Intersection of FSAs

Recall from previous lectures how the FSA are intersected. (Note: This is NOT the way to deal with Büchi automata!) The following FSA automaton $\mathcal{A}$ is the intersection of $\mathcal{A}_1$ and $\mathcal{A}_2$, when they are seen as FSAs.



# Example: Intersection of BAs

The following Büchi automaton $\mathcal{A}$ is the intersection $\mathcal{A} = \mathcal{A}_1 \cap \mathcal{A}_2$.



## From Kripke Structure to Büchi

Actually, we were careful to define the mapping from Kripke structures to finite state automata in such a way that it will work also without modifications with Büchi automata.

## From Kripke Structure to Büchi cnt.

Let $M = (S, s^0, R, L)$ be a Kripke structure over a set of atomic propositions $AP$. Define a Büchi automaton $\mathcal{A}_M = (\Sigma, S_M, S_M^0, \Delta_M, F_M)$, where

- $\Sigma = 2^{AP}$,
- $S_M = S \cup \{s^i\}$,
- $S_M^0 = \{s^i\}$,
- For all $s, s' \in S_M, a \in \Sigma : (s, a, s') \in \Delta_M$ iff $L(s') = a$ and $(((s, s') \in R)$ or $(s = s^i$ and $s' = s^0))$;
- and $F_M = S_M$.

- The Büchi automaton $\mathcal{A}_M$ accepts exactly those infinite sequences of labellings which correspond to infinite paths of the Kripke structure starting from some initial state.
- We will later show how given an $LTL$ formula $f$, we can create a Büchi automaton $\mathcal{A}_f$ which accepts exactly all the infinite sequences of valuations which satisfy $f$.
- In model checking we actually negate the property $f$ first, and then create a Büchi automaton $\mathcal{A}_{\neg f}$. This automaton accepts all violations of the property $f$.

- If we have been given $\mathcal{A}_{\neg f}$, it holds that $M \models f$ iff $\mathcal{L}\left(\mathcal{A}_M \times \mathcal{A}_{\neg f}\right) = \emptyset$.
- In other words: if no path of the Kripke structure is a model of the complement of the specification, then all paths of the Kripke structure are models of the specification.

# Kripke Product

- We'll now show a small trick, using which $\mathcal{A}_M \times \mathcal{A}_{\neg f}$ can be replaced by a slightly smaller automaton, which we call a Kripke product, and denote by $\mathcal{A}_M \otimes \mathcal{A}_{\neg f}$.
- In the special case $F_1 = S_1$ we can actually use a simpler product construction, we denote it by $\mathcal{A}_1 \otimes \mathcal{A}_2$:
- Let $\mathcal{A}_1 = (\Sigma, S_1, S_1^0, \Delta_1, F_1)$,$\mathcal{A}_2 = (\Sigma, S_2, S_2^0, \Delta_2, F_2)$ be two Büchi automata, such that for automaton $\mathcal{A}_1$ it holds that $F_1 = S_1$. (Note that this is the case when $\mathcal{A}_1$ is generated from a Kripke structure.)

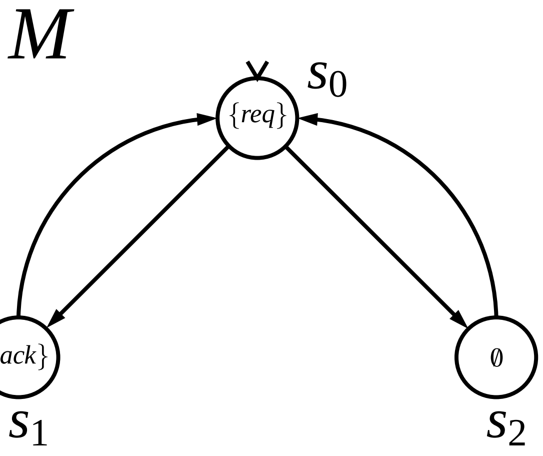
**Definition 4** We define the *Kripke product* automaton to be $\mathcal{A} = (\Sigma, S, S^0, \Delta, F)$, where:

- $S = S_1 \times S_2$,
- $S^0 = S_1^0 \times S_2^0$,
- for all $s, s' \in S_1, t, t' \in S_2, a \in \Sigma$:
  $((s,t), a, (s', t')) \in \Delta$ iff $(s, a, s') \in \Delta_1$ and $(t, a, t') \in \Delta_2$; and
- $F = S_1 \times F_2$.

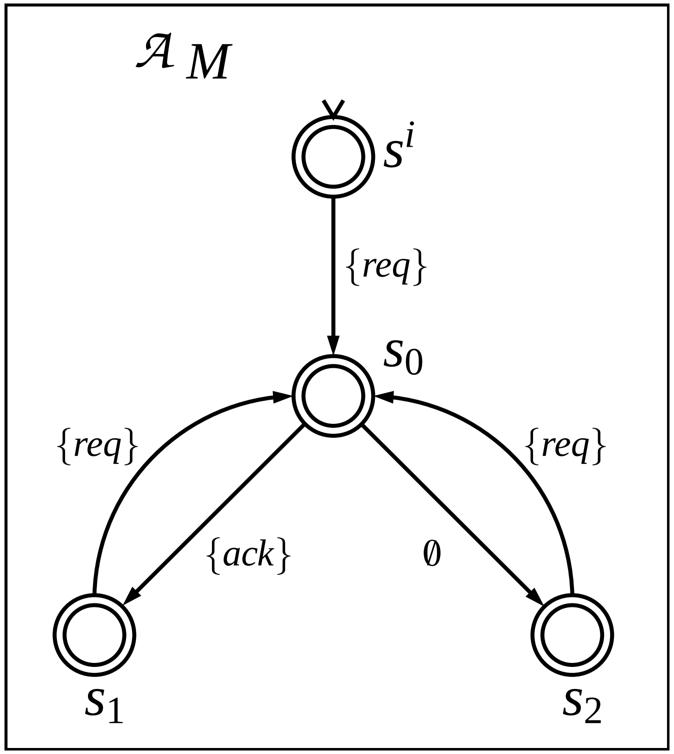
- Now for the Kripke product Büchi automaton $\mathcal{A}$ (also denoted by $\mathcal{A}_1 \otimes \mathcal{A}_2$) it holds that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. (Side note: The above definition happens to be equivalent to the FSA $\mathcal{A} = \mathcal{A}_1 \cap \mathcal{A}_2$ operation because $S_1 = F_1$!)
- Now if $\mathcal{A}_1$ is a Kripke structure automaton $\mathcal{A}_M$, it fulfills the property above, and thus this *Kripke product* construction can be used instead. (It has half as many states.)

# Example: LTL Model Checking

Assume we want to check whether $M \models f$, where $f = \Box(req \Rightarrow (\Diamond ack))$ for the Kripke structure $M$ below. This can be solved by checking whether for the Kripke product automaton $\mathcal{P} = \mathcal{A}_M \otimes \mathcal{A}_{\neg f}$ it holds that $\mathcal{L}(\mathcal{P}) = \emptyset$. If so, then $M \models f$, otherwise $M \not\models f$.

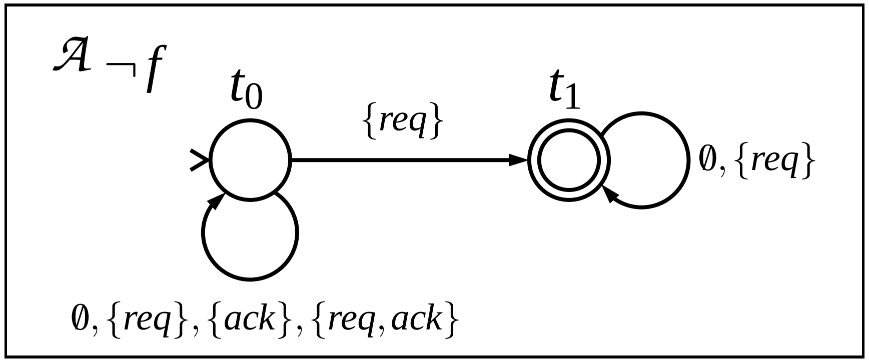


## Büchi Automaton $\mathcal{A}_M$

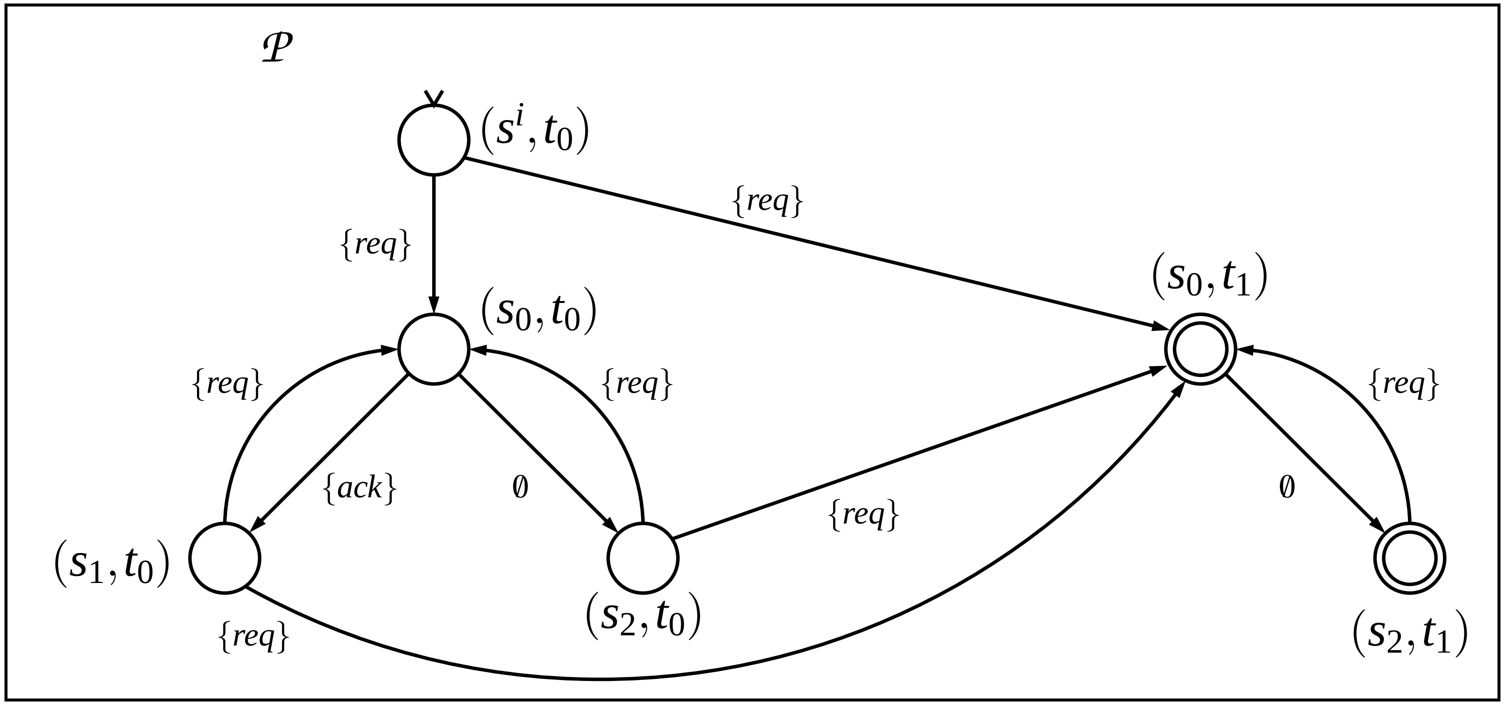


## Büchi Automaton $\mathcal{A}_{\neg f}$

Now it is easy to see that $\neg f = \Diamond(req \land (\Box \neg ack))$.



## Büchi Automaton $\mathcal{P}$



## Büchi Automaton $\mathcal{P}$ is Non-empty!

- It is easy to see that the Büchi automaton $\mathcal{P}$ has accepting runs. An example is the run $r = (s^i, t_0), (s_0, t_1), (s_2, t_1), (s_0, t_1), (s_2, t_1), \ldots$.
- Now $\mathit{inf}(r) = \{(s_0, t_1), (s_2, t_1)\}$, and thus the run $r$ is accepting.

- The word accepted by $r$ is:
  $w = \{req\}, \emptyset, \{req\}, \emptyset, \{req\}, \emptyset, \ldots$.
- The path $\pi$ the run $r$ corresponds to can be obtained from $r$ by projecting $r$ on the first component, and dropping the special state $s^i$ from the beginning, i.e., $\pi = s_0, s_2, s_0, s_2, \ldots$.
- We also get that $\pi \models \neg f$, and finally that $M \not\models f$.

## Emptiness Checking for BA

- The following “nested depth-first search” algorithm can be used to check a Büchi automaton for emptiness.
- It uses a hash table to check whether a state $s$ has already been visited by the algorithm. A new state can be stored into this table using subroutine “hash($s$)”.
- For efficiency each hash table entry contains (only) two bits of additional information, both initialized to zero value.

## Emptiness Checking for BA cnt.

- To manipulate these bits, there are the following subroutines. The subroutine “addstack1($s$)” turns the first bit to one, the subroutine “removestack1($s$)” clears the first bit, and the subroutine “instack1($s$)” returns “True” iff the first bit is set.
- The subroutine “flag($s$)” turns the second bit to one, and the subroutine “flagged($s$)” returns “True” iff the second bit is set.

**Algorithm 1** The top-level nested DFS algorithm

```
procedure emptiness
    for all $s \in S^0$ do
        dfs1($s$);
    terminate(False); // Automaton is empty
end procedure
```

**Algorithm 2** The dfs1 subroutine

```
procedure dfs1($s$)
    local state $s'$;
    hash($s$);
    addstack1($s$);
    // $((s, a, s') \in \Delta$ for some $a \in \Sigma)$
    for all successors $s'$ of $s$ do
        if $s'$ is not in the hash table then dfs1($s'$);
    if $s$ is an accepting state then dfs2($s$); // $(s \in F)$
    removestack1($s$);
end procedure
```

```
**Algorithm 3** The dfs2 subroutine
procedure dfs2($s$)
    local state $s'$;
    flag($s$);
    // $((s,a,s') \in \Delta$ for some $a \in \Sigma)$
    for all successors $s'$ of $s$ do
        // Accepting run through $s'$ found?
        if instack1($s'$) then terminate(True);
        else if not flagged($s'$) then dfs2($s'$);
        end if;
end procedure
```

- Actually DFS search order is needed for correctness only in the subroutine “dfs1($s$)”. (Using DFS there is vital for correctness!)
- The subroutine “dfs2($s$)” can actually be implemented using any search order (for example BFS).
- The above emptiness checking algorithm “nested depth first search” is what is implemented in the $LTL$ model checker SPIN.

## MSCCs

- We define a strongly connected component $C$ of a directed graph to be a set of nodes $C \subseteq S$, in which for all pairs of distinct states $s, s' \in C$ it holds that: $s'$ can be reached from $s$ and $s$ can be reached from $s'$.
- A strongly connected component $C$ is called maximal, if no strongly connected component $C' \subseteq S$ exists, such that $C \subset C'$.

- A maximal strongly connected component is called non-trivial iff: (i) $|C| > 1$, or (ii) there exists $s \in C$ such that there is an edge in the graph from $s$ back to $s$.

# Emptiness Checking with MSCCs

- Another way of checking the non-emptiness of $\mathcal{L}(\mathcal{A})$ is to compute the maximal strongly connected components (MSCCs) of the Büchi automaton, and check whether some non-trivial maximal strongly connected component $C$ reachable from some initial state $s \in S^0$ contains an accepting state ($C \cap F \neq \emptyset$). If so, the language is non-empty. Otherwise, the language is empty.

- Also this emptiness checking approach can be implemented with a linear time algorithm, e.g. by using the Tarjan's algorithm for computing the MSCCs. (Compute the reachable MSCCs and check whether any non-trivial MSCC contains an accepting state.)

## Deterministic Büchi Automata

- The definition of determinism for Büchi automata is identical to the FSA case.
- Unlike finite state automata, Büchi automata are not expressively complete when deterministic. In other words, there are languages accepted by non-deterministic Büchi automata, which no deterministic Büchi automaton accepts. An example of such a language is $(a+b)^*b^\omega$. (A finite number of $a$ symbols with finitely many occurrences of $b$ symbols between any two $a$'s followed by an infinite sequence of $b$ symbols.)

## Deterministic Büchi Automata cnt.

- Also note that $LTL$ requires non-deterministic automata to be expressed, the language above is effectively the same as the requirement expressed by the $LTL$ formula $\Diamond\Box b$.

## Complementing Büchi Automata

- A complementation procedure for Büchi automata exists, however, it is very different from finite state automata, as the normal determinization construction can not be used. In fact, we have the following result:

**Theorem 5** Let $\mathcal{A}$ be any (non-deterministic) Büchi automaton with $n$ states. Then in the worst case the smallest Büchi automaton $\mathcal{A}'$, such that $\mathcal{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$ will have $\Omega(n!)$ $(= 2^{\Omega(n \log n)})$ states.

## Complementing Büchi Automata cnt.

- This blow-up is in practice much worse than the blow-up for finite state automata complementation. (For example, while $5! = 120$ and $2^5 = 32$, factorial grows much faster: $10! = 3628800$, while $2^{10} = 1024$.)
- There are several different ways to complement Büchi automata matching the lower bound. Thankfully in (basic) model checking complementation of Büchi automata is not needed.