COMP6210 Automated Software Verification

Automata-Based Verification for LTL

Pavel Naumov

Intended Learning Outcomes

By the end of this lecture, you will be able to

- define a Büchi automaton
- construct Buchi automata for simple LTL formulas
- explain how Buchi automata are used in model-checking LTL formulas

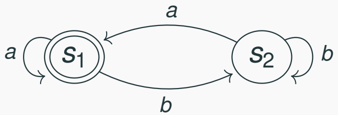
Outline

Büchi Automata

LTL Model Checking

different btw fa and ts

- finite automata accept/reject *strings* over a given *alphabet*
- equivalent to regular expressions
- a finite automaton over the alphabet $\{a, b\}$:



  - the initial state is s_1
  - the final state is s_1
  - the string *abba* is accepted
  - the string *bab* is rejected

Recap on Finite Automata (2)

Let Σ be a finite alphabet (consisting symbols that we will use in strings/words over Σ).

A **finite automaton** $\mathbb{A}$ consists of:

- a finite set of states: s
- a finite set of transitions between states, labelled by symbols from Σ: $s \xrightarrow{a} t$
- a subset of initial states: $\longrightarrow s$
- a subset of final/accepting states: s

Recap on Finite Automata (3)

- A **run** of an automaton on a finite word $a_1 \dots a_n$ is a finite path through the automaton starting in an initial state, with transitions labelled by $a_1, \dots, a_n$ (in this order):

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$$

- An automaton $\mathbb{A}$ **accepts** a finite word $a_1 \dots a_n$ if there exists a run of $\mathbb{A}$ on $a_1, \dots, a_n$ which ends in an accepting state.

- The **language** of an automaton $\mathbb{A}$, denoted $L(\mathbb{A})$, consists of all the words it accepts.

Recap on Finite Automata (3)

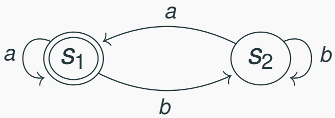
- A **run** of an automaton on a finite word $a_1 \dots a_n$ is a finite path through the automaton starting in an initial state, with transitions labelled by $a_1, \dots, a_n$ (in this order):

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$$

- An automaton $\mathbb{A}$ **accepts** a finite word $a_1 \dots a_n$ if there exists a run of $\mathbb{A}$ on $a_1, \dots, a_n$ which ends in an accepting state.

- The **language** of an automaton $\mathbb{A}$, denoted $L(\mathbb{A})$, consists of all the words it accepts.

Recap on Finite Automata (4)



- some runs of this automaton:

$$s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2 \xrightarrow{a} s_1$$

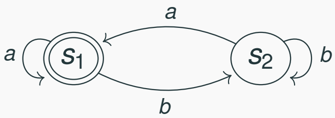
$$s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2 \xrightarrow{a} s_1$$

- some words accepted by this automaton:

  ϵ (the empty word), *a*, *ba*, *bba*, *baba*, …

- the language of this automaton consists of:
  - the empty word,
  - all non-empty words that end with an *a*.

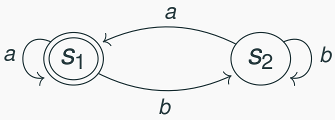
Finite Automata over Infinite Words (Büchi Automata)



(The initial state is s_1.)

- An **infinite run** is an infinite path through the automaton, starting in an initial state.
  - e.g. $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$
- An automaton **accepts** an infinite word $a_1 a_2 a_3 \ldots$ if there exists a run of the automaton labelled by $a_1, a_2, \ldots$, which passes through an accepting state infinitely often. We call such a run accepting.

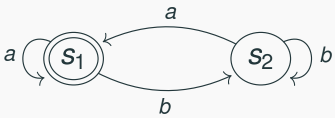
Finite Automata over Infinite Words (Büchi Automata)



(The initial state is s_1.)

- An **infinite run** is an infinite path through the automaton, starting in an initial state.
  - e.g. $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$
- An automaton **accepts** an infinite word $a_1 a_2 a_3 \ldots$ if there exists a run of the automaton labelled by $a_1, a_2, \ldots$, which passes through an accepting state infinitely often. We call such a run accepting.

Example



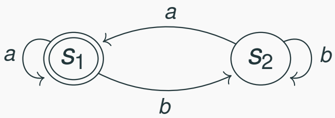
(The only initial state and only final state is s_1.)

The run $s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} \dots$ is accepting.

The run $s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2 \xrightarrow{b} \dots$ is not accepting.

The language of this Büchi automaton consists of all infinite words containing infinitely many *a* symbols.

Example



(The only initial state and only final state is s_1.)

The run $s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} \ldots$ is accepting.

The run $s_1 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{b} s_2 \xrightarrow{b} \ldots$ is not accepting.

The language of this Büchi automaton consists of all infinite words containing infinitely many a symbols.

Some Properties of Büchi Automata

If $\mathbb{A}$ and $\mathbb{B}$ are automata, then there exist:

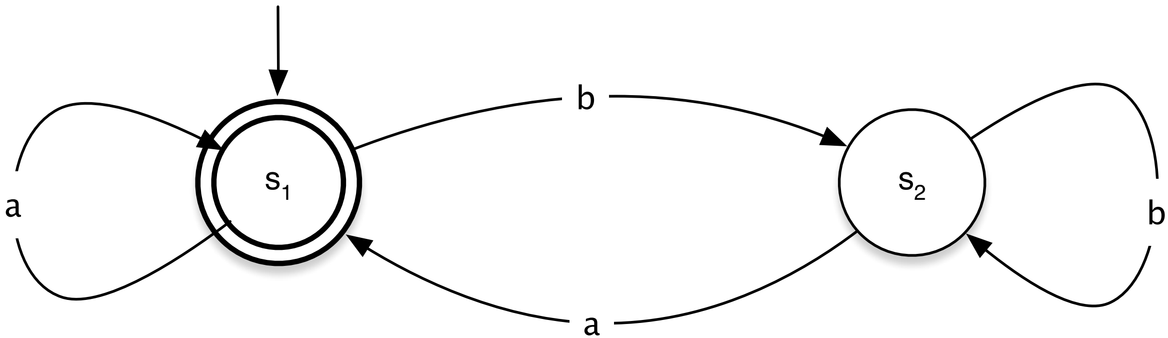
- an automaton $\mathbb{A} \cap \mathbb{B}$ (the product automaton) such that
  $$L(\mathbb{A} \cap \mathbb{B}) = L(\mathbb{A}) \cap L(\mathbb{B}).$$
- an automaton $\overline{\mathbb{A}}$ (the complement automaton) such that $L(\overline{\mathbb{A}})$ contains exactly those infinite words which are not in $L(\mathbb{A})$.

Some Properties of Büchi Automata

If $\mathbb{A}$ and $\mathbb{B}$ are automata, then there exist:

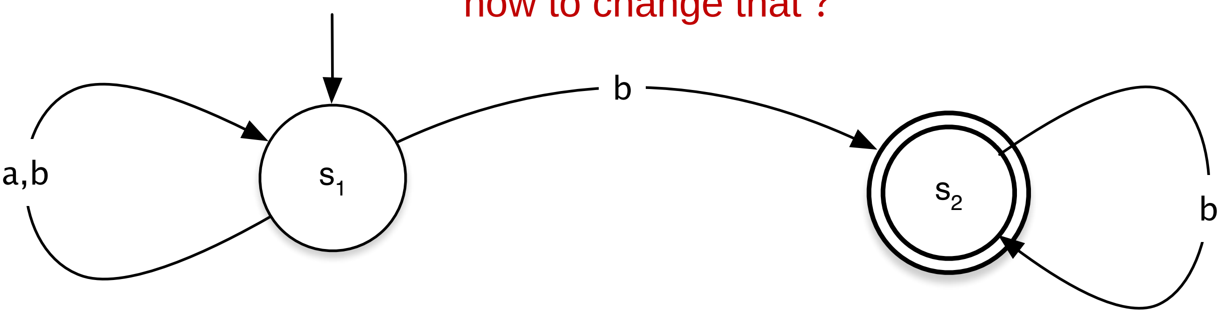
- an automaton $\mathbb{A} \cap \mathbb{B}$ (the product automaton) such that
$$L(\mathbb{A} \cap \mathbb{B}) = L(\mathbb{A}) \cap L(\mathbb{B}).$$
- an automaton $\overline{\mathbb{A}}$ (the complement automaton) such that $L(\overline{\mathbb{A}})$ contains exactly those infinite words which are not in $L(\mathbb{A})$.

Buchi Automaton Complement Construction



accepts infinite words that have infinitely many symbols a

how to change that ?



无限个a + 无限个b ?

accepts infinite words that have finitely many symbols a

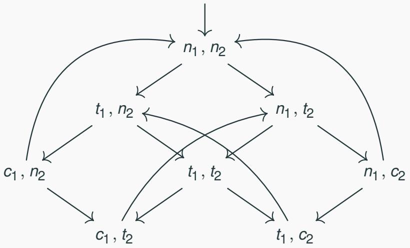
Outline

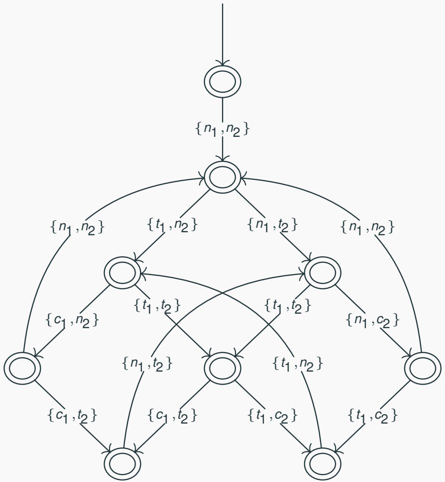
Büchi Automata

LTL Model Checking

Model-Checking using Automata (1)

Transition systems can be transformed into Büchi automata, e.g.





Model-Checking using Automata (2)

- The states of the automaton correspond to states of the transition system, plus one additional initial state.
- There is one transition from the new (and only!) initial state to all the states corresponding to initial states in the transition system.
- All other transitions of the automaton correspond to transitions in the transition system.
- The alphabet of the automaton is the set of all subsets of *Prop*, that is $\mathcal{P}(\mathit{Prop})$.
- The labels on automaton transitions are inherited from the *target states* in the transition system.
- All automaton states are accepting. (So all runs will be accepting!)

Model-Checking using Automata (3)

Question: What are the words accepted by the resulting automaton?

Answer: Exactly those infinite sequences of sets of atomic propositions that occur along computation paths through the transition system !

For example:

- atomic propositions: $Prop = \{n_1, n_2, t_1, t_2, c_1, c_2\}$
- $\{n_1, n_2\} \to \{t_1, n_2\} \to \{c_1, n_2\} \to \{n_1, n_2\} \to \dots$ occurs along a path through the transition system
- $\{n_1, n_2\}, \{t_1, n_2\}, \{c_1, n_2\}, \{n_1, n_2\}, \dots$ is accepted by the automaton

Intuition: runs of the automaton correspond to paths through the transition system.

Model-Checking using Automata (3)

Question: What are the words accepted by the resulting automaton?

Answer: Exactly those infinite sequences of sets of atomic propositions that occur along computation paths through the transition system !

For example:

- atomic propositions: $Prop = \{n_1, n_2, t_1, t_2, c_1, c_2\}$
- $\{n_1, n_2\} \to \{t_1, n_2\} \to \{c_1, n_2\} \to \{n_1, n_2\} \to \ldots$ occurs along a path through the transition system
- $\{n_1, n_2\}, \{t_1, n_2\}, \{c_1, n_2\}, \{n_1, n_2\}, \ldots$ is accepted by the automaton

Intuition: runs of the automaton correspond to paths through the transition system.

Model-Checking using Automata (3)

Question: What are the words accepted by the resulting automaton?

Answer: Exactly those infinite sequences of sets of atomic propositions that occur along computation paths through the transition system !

For example:

- atomic propositions: $Prop = \{n_1, n_2, t_1, t_2, c_1, c_2\}$
- $\{n_1, n_2\} \to \{t_1, n_2\} \to \{c_1, n_2\} \to \{n_1, n_2\} \to \ldots$ occurs along a path through the transition system
- $\{n_1, n_2\}, \{t_1, n_2\}, \{c_1, n_2\}, \{n_1, n_2\}, \ldots$ is accepted by the automaton

Intuition: runs of the automaton correspond to paths through the transition system.

Model-Checking using Automata (4)

- LTL formula involving *Prop* can *also* be transformed into an automaton over the alphabet $\mathcal{P}(\mathit{Prop})$.

  Intuition:
  - infinite words over $\mathcal{P}(\mathit{Prop})$ describe sequences of sets of atomic propositions
  - the resulting automaton should accept an infinite word precisely when the corresponding sequence satisfies the given formula
- since the model and the specification are both automata, we can use the theory of Büchi automata to do model-checking !

Model-Checking using Automata (4)

- LTL formula involving *Prop* can *also* be transformed into an automaton over the alphabet $\mathcal{P}(Prop)$.

  Intuition:
  - infinite words over $\mathcal{P}(Prop)$ describe sequences of sets of atomic propositions
  - the resulting automaton should accept an infinite word precisely when the corresponding sequence satisfies the given formula

- since the model and the specification are both automata, we can use the theory of Büchi automata to do model-checking !

From LTL to Automata (Mutual Exclusion Property)

$\mathbf{A}\ \mathbf{G}\,\neg(c_1 \wedge c_2)$ becomes

where:

- $\bigcirc \xrightarrow{c_1 \wedge c_2} \bigcirc$ stands for all arcs labelled by both c_1 and c_2:

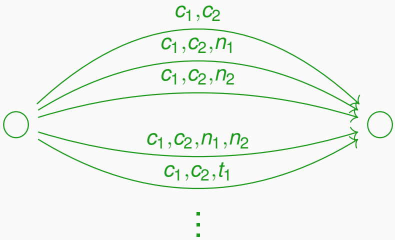
- $\bigcirc \xrightarrow{\texttt{tt}} \bigcirc$ stands for *all* arcs labelled with subsets of *Prop*
- $\bigcirc \xrightarrow{\neg(c_1 \wedge c_2)} \bigcirc$ stands for *all* arcs labelled with subsets of *Prop* not containing both c_1 and c_2

From LTL to Automata (Mutual Exclusion Property)

$\mathbf{A}\ \mathbf{G}\, \neg(c_1 \wedge c_2)$ becomes

where:

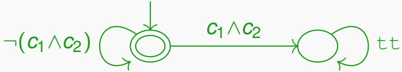
- $\bigcirc \xrightarrow{c_1 \wedge c_2} \bigcirc$ stands for all arcs labelled by both c_1 and c_2:



- $\bigcirc \xrightarrow{\texttt{tt}} \bigcirc$ stands for *all* arcs labelled with subsets of *Prop*
- $\bigcirc \xrightarrow{\neg(c_1 \wedge c_2)} \bigcirc$ stands for *all* arcs labelled with subsets of *Prop* not containing both c_1 and c_2

From LTL to Automata

- mutual exclusion property: $\mathbf{A}\ \mathbf{G}\, \neg(c_1 \wedge c_2)$



This automaton accepts all infinite words not containing both c_1 and c_2 at the same time (i.e. within the same "symbol").

- liveness property: $\mathbf{A}\ \mathbf{F}\, c_1$:

This automaton accepts all infinite words eventually containing c_1 as part of a "symbol".

- mutual exclusion property: $\mathbf{A}\ \mathbf{G}\ \neg(c_1 \wedge c_2)$

This automaton accepts all infinite words not containing both c_1 and c_2 at the same time (i.e. within the same "symbol").

- liveness property: $\mathbf{A}\ \mathbf{F}\ c_1$:

other examples

This automaton accepts all infinite words eventually containing c_1 as part of a "symbol".

Model Checking Using Automata (5)

- model captured by Büchi automaton $\mathbb{A}$
- specification (LTL formula) captured by Büchi automaton $\mathbb{S}$
- **Key observation:**

  model satisfies the specification iff $L(\mathbb{A}) \subseteq L(\mathbb{S})$!

  (any behaviour in the model satisfies the specification)
- thus, checking property "$\mathbb{S}$" on model "$\mathbb{A}$" reduces to checking the language inclusion $L(\mathbb{A}) \subseteq L(\mathbb{S})$
- $L(\mathbb{A}) \subseteq L(\mathbb{S})$ can, in turn, be checked by checking:

  $$L(\mathbb{A}) \cap \overline{L(\mathbb{S})} = \emptyset$$

  or equivalently:

  $$L(\mathbb{A} \cap \overline{\mathbb{S}}) = \emptyset$$

Model Checking Using Automata (5)

- model captured by Büchi automaton $\mathbb{A}$
- specification (LTL formula) captured by Büchi automaton $\mathbb{S}$
- **Key observation:**

  model satisfies the specification iff $L(\mathbb{A}) \subseteq L(\mathbb{S})$!

  (any behaviour in the model satisfies the specification)
- thus, checking property "$\mathbb{S}$" on model "$\mathbb{A}$" reduces to checking the language inclusion $L(\mathbb{A}) \subseteq L(\mathbb{S})$
- $L(\mathbb{A}) \subseteq L(\mathbb{S})$ can, in turn, be checked by checking:

  $$L(\mathbb{A}) \cap \overline{L(\mathbb{S})} = \emptyset$$

  or equivalently:

  $$L(\mathbb{A} \cap \overline{\mathbb{S}}) = \emptyset$$

Model Checking Using Automata (5)

- model captured by Büchi automaton $\mathbb{A}$
- specification (LTL formula) captured by Büchi automaton $\mathbb{S}$
- **Key observation:**

  model satisfies the specification iff $L(\mathbb{A}) \subseteq L(\mathbb{S})$!

  (any behaviour in the model satisfies the specification)
- thus, checking property "$\mathbb{S}$" on model "$\mathbb{A}$" reduces to checking the language inclusion $L(\mathbb{A}) \subseteq L(\mathbb{S})$
- $L(\mathbb{A}) \subseteq L(\mathbb{S})$ can, in turn, be checked by checking:

  $$L(\mathbb{A}) \cap \overline{L(\mathbb{S})} = \emptyset$$

  or equivalently:

  $$L(\mathbb{A} \cap \overline{\mathbb{S}}) = \emptyset$$

Model Checking Using Automata (5)

- model captured by Büchi automaton $\mathbb{A}$
- specification (LTL formula) captured by Büchi automaton $\mathbb{S}$
- **Key observation:**

  model satisfies the specification iff $L(\mathbb{A}) \subseteq L(\mathbb{S})$!

  (any behaviour in the model satisfies the specification)
- thus, checking property "$\mathbb{S}$" on model "$\mathbb{A}$" reduces to checking the language inclusion $L(\mathbb{A}) \subseteq L(\mathbb{S})$
- $L(\mathbb{A}) \subseteq L(\mathbb{S})$ can, in turn, be checked by checking:

  $$L(\mathbb{A}) \cap \overline{L(\mathbb{S})} = \emptyset$$

  or equivalently:

  $$L(\mathbb{A} \cap \overline{\mathbb{S}}) = \emptyset$$

Model Checking Using Automata (6)

Need to compute automaton $\mathbb{A} \cap \overline{\mathbb{S}}$ ...

- computing $\mathbb{A} \cap \mathbb{B}$ is relatively easy (polynomial complexity) ...
- ...but the complexity of computing $\overline{\mathbb{S}}$ is exponential in the number of states if $\mathbb{S}$ is *non-deterministic* !
  - what if $\mathbb{S}$ is *deterministic* ?
  - **Note:** not any non-deterministic Büchi automaton has an equivalent deterministic one !
- better to directly generate automaton for the negation of the LTL property to check !

Model Checking Using Automata (6)

Need to compute automaton $\mathbb{A} \cap \overline{\mathbb{S}}$ . . .

- computing $\mathbb{A} \cap \mathbb{B}$ is relatively easy (polynomial complexity) . . .
- . . . but the complexity of computing $\overline{\mathbb{S}}$ is exponential in the number of states if $\mathbb{S}$ is *non-deterministic* !
  - what if $\mathbb{S}$ is *deterministic* ?
  - **Note:** not any non-deterministic Büchi automaton has an equivalent deterministic one !
- better to directly generate automaton for the negation of the LTL property to check !

Model Checking Using Automata (6)

Need to compute automaton $\mathbb{A} \cap \overline{\mathbb{S}}$ ...

- computing $\mathbb{A} \cap \mathbb{B}$ is relatively easy (polynomial complexity) ...
- ...but the complexity of computing $\overline{\mathbb{S}}$ is exponential in the number of states if $\mathbb{S}$ is *non-deterministic* !
  - what if $\mathbb{S}$ is *deterministic* ?
  - **Note:** not any non-deterministic Büchi automaton has an equivalent deterministic one !
- better to directly generate automaton for the negation of the LTL property to check !

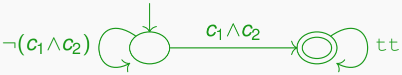
Model Checking Using Automata (6)

Need to compute automaton $\mathbb{A} \cap \overline{\mathbb{S}}$ ...

- computing $\mathbb{A} \cap \mathbb{B}$ is relatively easy (polynomial complexity) ...
- ... but the complexity of computing $\overline{\mathbb{S}}$ is exponential in the number of states if $\mathbb{S}$ is *non-deterministic* !
  - what if $\mathbb{S}$ is *deterministic* ?
  - **Note:** not any non-deterministic Büchi automaton has an equivalent deterministic one !
- better to directly generate automaton for the negation of the LTL property to check !

Example 1

- automaton for negation of mutual exclusion property:

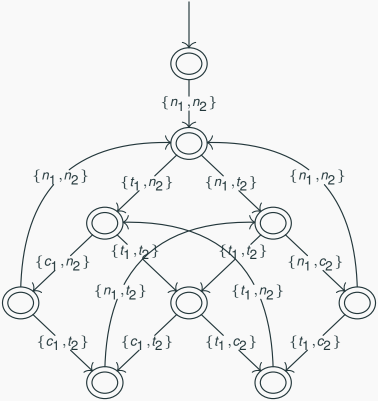


- automaton for model of mutual exclusion:

Example 1

- automaton for negation of mutual exclusion property:

- automaton for model of mutual exclusion:



combine

Construction of the Product Automaton

- the states of the product automaton are given by *pairs of states* of the two automata,

  Note: this is only possible if all the states of one of the automata are accepting.

- $(s_1, s_2) \xrightarrow{a} (s'_1, s'_2)$ precisely when $(s_1) \xrightarrow{a} (s'_1)$ and $(s_2) \xrightarrow{a} (s'_2)$

- (s_1, s_2) is an initial state precisely when both s_1 and s_2 are initial states,

- (s_1, s_2) is an accepting state precisely when both s_1 and s_2 are accepting states.

Construction of the Product Automaton

- the states of the product automaton are given by *pairs of states* of the two automata,

  Note: this is only possible if all the states of one of the automata are accepting.

- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ precisely when $(s_1) \xrightarrow{a} (s_1')$ and $(s_2) \xrightarrow{a} (s_2')$

- (s_1, s_2) is an initial state precisely when both s_1 and s_2 are initial states,

- (s_1, s_2) is an accepting state precisely when both s_1 and s_2 are accepting states.

Construction of the Product Automaton

- the states of the product automaton are given by *pairs of states* of the two automata,

  Note: this is only possible if all the states of one of the automata are accepting.

- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ precisely when $(s_1) \xrightarrow{a} (s_1')$ and $(s_2) \xrightarrow{a} (s_2')$

- (s_1, s_2) is an initial state precisely when both s_1 and s_2 are initial states,

- (s_1, s_2) is an accepting state precisely when both s_1 and s_2 are accepting states.

Construction of the Product Automaton

- the states of the product automaton are given by *pairs of states* of the two automata,

  Note: this is only possible if all the states of one of the automata are accepting.

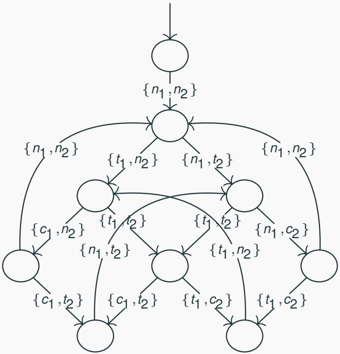
- $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ precisely when $s_1 \xrightarrow{a} s_1'$ and $s_2 \xrightarrow{a} s_2'$

- (s_1, s_2) is an initial state precisely when both s_1 and s_2 are initial states,

- (s_1, s_2) is an accepting state precisely when both s_1 and s_2 are accepting states.

Example 1 (Cont'd)

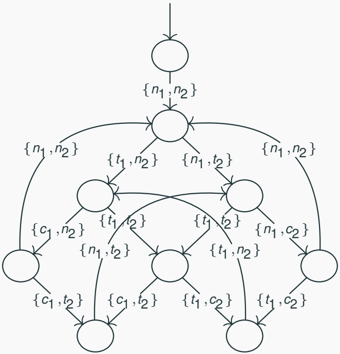
Product automaton has no accepting states, hence does not accept *any* infinite word:



Since the language accepted by the product automaton is **empty**, it follows that the mutual exclusion property **A G** $\neg(c_1 \wedge c_2)$ **holds** in the original transition system.

Example 1 (Cont'd)

Product automaton has no accepting states, hence does not accept *any* infinite word:



Since the language accepted by the product automaton is **empty**, it follows that the mutual exclusion property **A G** $\neg(c_1 \wedge c_2)$ **holds** in the original transition system.

Example 2

- automaton for negation of liveness property:

Note: this is not deterministic!

c1 never happen

buchi automaton id used for infinte path

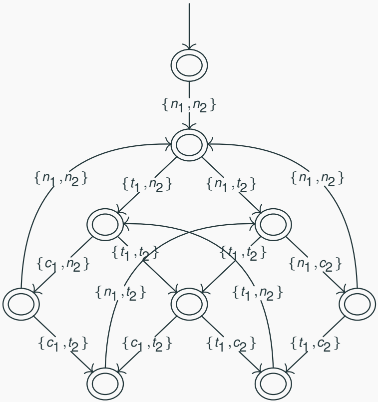
- automaton for model of mutual exclusion:

Example 2

- automaton for negation of liveness property:

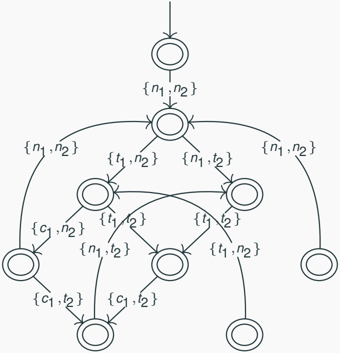
Note: this is not deterministic!

- automaton for model of mutual exclusion:



Example 2 (Cont'd)

Product automaton:

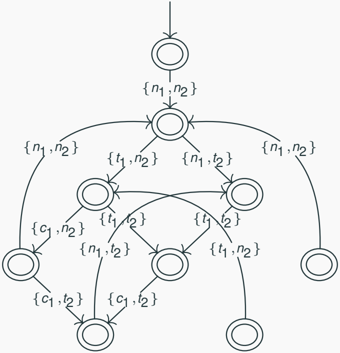


Since the language accepted by the product automaton is **not empty** (why?), it follows that the liveness property **A F** c_1 **does not hold** in the original transition system.

Any infinite word accepted by the above automaton produces a counterexample!

Example 2 (Cont'd)

Product automaton:

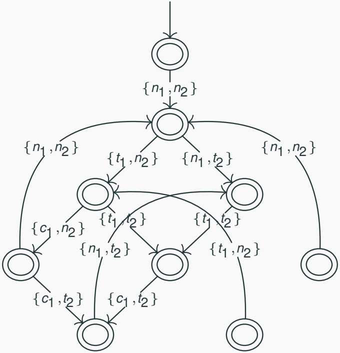


Since the language accepted by the product automaton is **not empty** (why?), it follows that the liveness property **A F** c_1 **does not hold** in the original transition system.

Any infinite word accepted by the above automaton produces a counterexample!

Example 2 (Cont'd)

Product automaton:



Since the language accepted by the product automaton is **not empty** (why?), it follows that the liveness property **A F** c_1 **does not hold** in the original transition system.

Any infinite word accepted by the above automaton produces a counterexample!

LTL Model-Checking Algorithm

- there is an algorithm for translating LTL formulas into Büchi automata (details in [Clarke et al.]),
- the complexity of the resulting model-checking algorithm is linear in the size of the model (and exponential in the length of the formula),
- fairness assumptions can also be described by LTL formulas.

Exercise

1. Construct an LTL formula and a Büchi automaton for the correctness property: *"In each execution, each process enters the critical section an infinite number of times."*
2. Check, by manually applying the model checking algorithm, whether this property holds in the model on slide 10.

The SPIN Model Checker

- example of explicit-state model checker
- uses Promela modelling language to describe (concurrent/distributed) software systems
- uses LTL as main mechanism for specifying correctness ...
- ... but annotations to Promela models also used for simple safety/liveness properties
- can handle systems with millions of states

Benefits and Limitations of Explicit State Model Checking

Benefits:

- automatic
- exhaustive
- produces counter-examples

Limitations:

- state explosion problem – partially addressed by symbolic model checking (our next topic)
- only works for finite-state systems