NSWI101: System Behaviour Models and Verification

2. Model Checking

Jan Kofroň

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of Distributed and Dependable Systems

Today

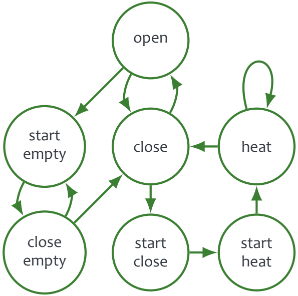
- Model checking
- Linear Temporal Logic (LTL)
- Büchi automata

Part I: Model Checking

MODEL CHECKING

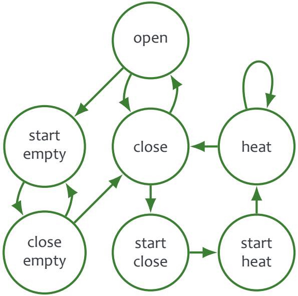
Model checking is process of determining whether given model M satisfies given property φ ($M \models \varphi$)

- In its basic form realized as traversal of finite graph
- Evaluates validity of property at given (initial) state
- Linear in size of graph (system model) and varying complexity in size of property (usually negligible)



System model

Model checking

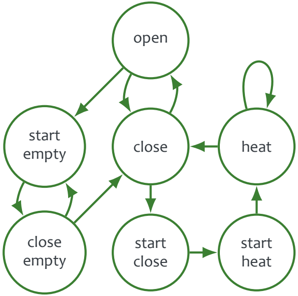


System model

AG (start → AF heat)

Property specification

MODEL CHECKING



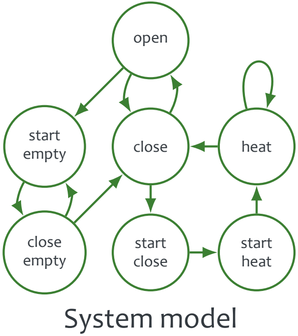
System model

Model Checker

AG (start → AF heat)

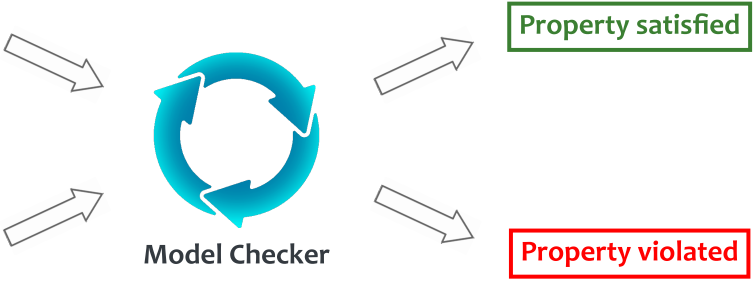
Property specification

MODEL CHECKING

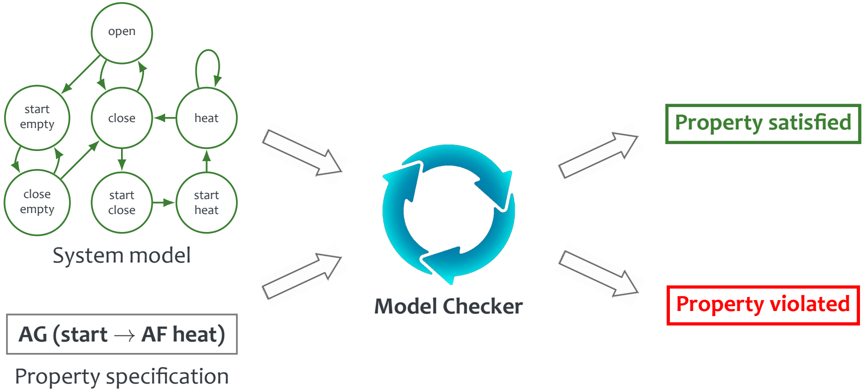


AG (start → AF heat)

Property specification



Model checking – Practical challenges



- Model construction
- Property expression
- State space explosion
- Error trace interpretation

State space explosion

The most serious issue of (explicit) model checking

- Model induces state space – combination of states of particular parts (behaviour of involved processes)
  - potentially exponential in size of model
- State space explosion = problem of too many states induced by the model
- Moore's law and algorithm advances can help to some extent:
  - 7 days in 1980 $\rightarrow$ 10 minutes in 1990 $\rightarrow$ 0.6 seconds in 2000 $\rightarrow$ ...
- However – explosive state growth in software inherently limits scalability

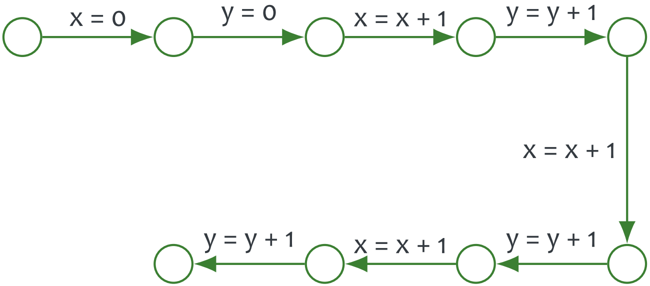
Modelling behaviour

Behaviour of system (program, hardware, computer system in general) can be captured in several ways:

- Labelled transition system
- Kripke structure
- Markov chain
- Timed automata
- ... and others

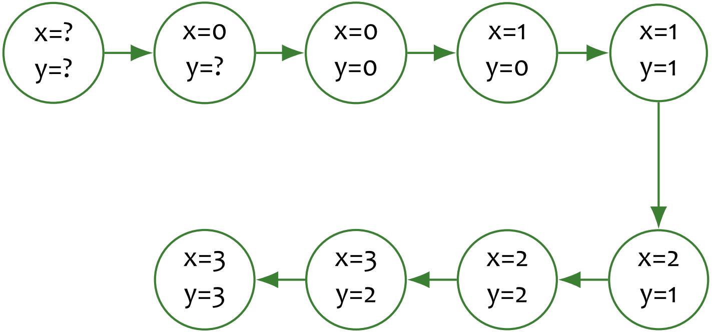
Reminder: Labelled Transition System

```
x:=0;
y:=0;

for i:=1 to 3 {
  x:=x+1;
  y:=y+1;
}
```



State Transition System

```
x:=0;
y:=0;

for i:=1 to 3 {
   x:=x+1;
   y:=y+1;
}
```



Kripke structure

- State transition system encoding only infinite paths
  - No finite paths allowed
- Suitable for systematic exploration
- Finite paths can be encoded, too, if desired

Kripke structure – Definition

For AP – set of atomic propositions (Boolean variables, constants, predicates), *Kripke structure* $M = (S, I, R, L)$ over AP is four-tuple:

- S – finite set of states
- $I \subseteq S$ – set of initial states
- $R \subseteq S \times S$ – transition relation such that R is left-total (for each state s there is a transition originating in it)
- $L : S \to 2^{AP}$ – labelling function

Creating Kripke structure of program

Each state of Kripke structure encodes a state of program, which includes:

- Program counters for all threads
- Values of all local variables for each thread
- Values of all global variables if present
- State (content) of heap memory
- System resources (opened files, database and network connections, ...)

There is transition $(s \to t) \in R$ if there is transition in program state corresponding to s transforming program state to one corresponding to t.

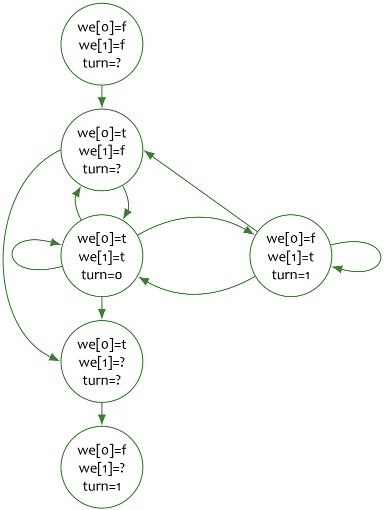
For final states add a self-loop to satisfy the left-total requirement from definition.

Example – Dekker's algorithm

```
bool wants_to_enter[0] = false;
bool wants_to_enter[1] = false;
int turn = ?;

void process (int: id) {
wants_to_enter[id] = true
   while (wants_to_enter[1-id]) {
      if (turn <> id) {
         wants_to_enter[id] = false
         while (turn <> id) ;
         wants_to_enter[id] = true
      }
   }

   // critical section

   turn = 1-id;
   wants_to_enter[id] = false;
}
```

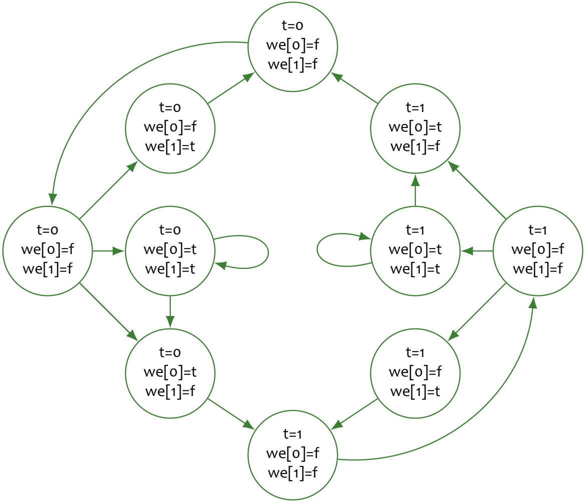
Example – Dekker's algorithm

```
bool wants_to_enter[0] = false;
bool wants_to_enter[1] = false;
int turn = ?;

void process (int: id) {
wants_to_enter[id] = true
   while (wants_to_enter[1-id]) {
      if (turn <> id) {
         wants_to_enter[id] = false
         while (turn <> id) ;
         wants_to_enter[id] = true
      }
   }

   // critical section

   turn = 1-id;
   wants_to_enter[id] = false;
}
```



Example – Dekker's algorithm

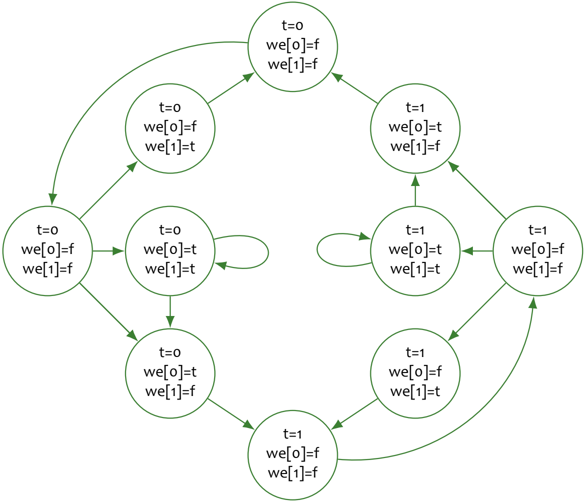
Composition of KS for two processes:



Example – Dekker's algorithm

Composition of KS for two processes:

Does the algorithm work correctly?
How to find out? I.e., how to formulate the property?

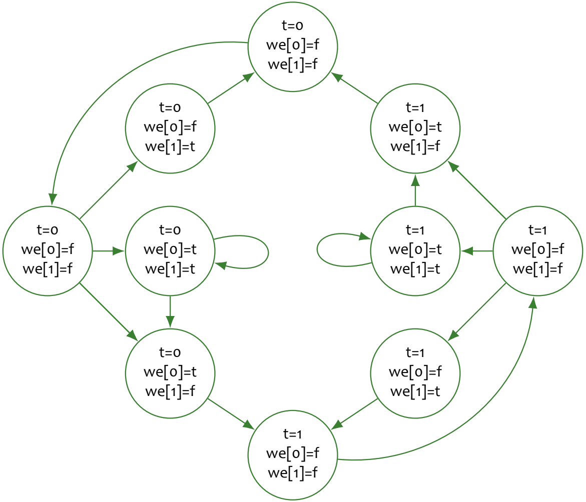


EXAMPLE – DEKKER'S ALGORITHM

Composition of KS for two processes:

Does the algorithm work correctly?
How to find out? I.e., how to formulate the property?

Can both processes get into critical section at the same time?



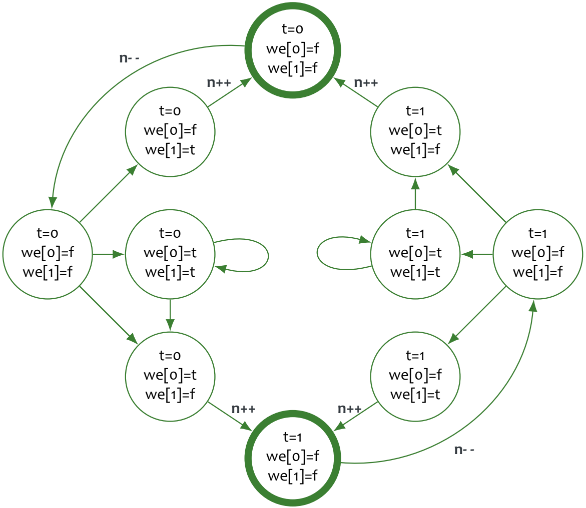
EXAMPLE – DEKKER'S ALGORITHM

Composition of KS for two processes:

Does the algorithm work correctly?
How to find out? I.e., how to formulate the property?

Can both processes get into critical section at the same time?

Artificial model variable can help.



EXAMPLE – DEKKER'S ALGORITHM

Composition of KS for two processes:

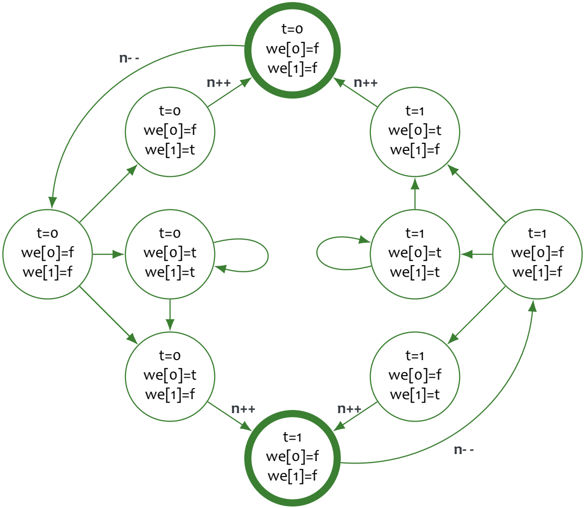
Does the algorithm work correctly? How to find out? I.e., how to formulate the property?

Can both processes get into critical section at the same time?

Artificial model variable can help.

Correctness property:

"*n* is always less than 2."



Part II: Linear Temporal Logic

Property specification

For property specification, **temporal logics** are usually used:

- Linear Time Logic (LTL)
- Computational Tree Logic (CTL)
- Probabilistic CTL (PCTL)
- Timed CTL – support for real-time properties
- …

Formal capturing of desired properties
"Temporal" – over time / along particular paths in model

Linear Temporal Logic

- Allows for expressing properties for any execution
- Particular paths in model considered one by one – details later
- Expressive enough for most common properties
- Efficient model checking algorithm linear in size of model and exponential in formula size

LTL SYNTAX

LTL syntax defined inductively, similarly to propositional logic:

Let *AP* be a finite set of Boolean variables (atomic propositions).
The set of LTL formulae over *AP* is defined as:

- If $p \in AP$ then p is LTL formula.
- If φ and ψ are LTL formulae then
  $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\mathrm{X}\,\varphi$, $\varphi\,\mathrm{U}\,\psi$, $\mathrm{F}\,\varphi$, $\varphi\,\mathrm{R}\,\psi$, and $\mathrm{G}\,\varphi$ are LTL formulae.

Negation, disjunction, X, and U are fundamental operators, others can be derived.

LTL SEMANTICS

Path in Kripke structure is infinite sequence $\pi = \pi_0, \pi_1, \pi_2, \ldots$ where for all $\forall i \geq 0.(\pi_i, \pi_{i+1}) \in R$

Let $M = (S, I, R, L)$ be Kripke structure and $\pi = \pi_0, \pi_1, \pi_2, \ldots$ be an infinite path in M. For an integer $i \geq 0$, π^i stands for i-th suffix of π: $\pi^i = \pi_i, \pi_{i+1}, \pi_{i+2}, \ldots$

Let M be Kripke structure, φ be LTL formula, π be path in M and s be state of M.

$M, \pi \models \varphi$: Path π from M satisfies φ
$M, s \models \varphi$: State s from M satisfies φ

- $M, s \models \varphi \leftrightarrow \forall \pi . \pi_0 = s : M, \pi \models \varphi$

LTL SEMANTICS

$$M, \pi \models p \quad \leftrightarrow \quad p \in L(\pi_0)$$

$$M, \pi \models \neg\varphi \quad \leftrightarrow \quad \neg(M, \pi \models \varphi)$$

$$M, \pi \models \varphi_1 \vee \varphi_2 \quad \leftrightarrow \quad M, \pi \models \varphi_1 \vee M, \pi \models \varphi_2$$

$$M, \pi \models \varphi_1 \wedge \varphi_2 \quad \leftrightarrow \quad M, \pi \models \varphi_1 \wedge M, \pi \models \varphi_2$$

$$M, \pi \models \mathrm{X}\,\varphi \quad \leftrightarrow \quad M, \pi^1 \models \varphi$$

$$M, \pi \models \mathrm{F}\,\varphi \quad \leftrightarrow \quad \exists i \geq 0 . M, \pi^i \models \varphi$$

$$M, \pi \models \mathrm{G}\,\varphi \quad \leftrightarrow \quad \forall i \geq 0 . M, \pi^i \models \varphi$$

$$M, \pi \models \varphi_1 \,\mathrm{U}\, \varphi_2 \quad \leftrightarrow \quad \exists i \geq 0 . M, \pi^i \models \varphi_2 \wedge \forall j . 0 \leq j < i \implies M, \pi^j \models \varphi_1$$

$$M, \pi \models \varphi_1 \,\mathrm{R}\, \varphi_2 \quad \leftrightarrow \quad (\mathrm{G}\,\varphi_2) \vee (\varphi_2 \,\mathrm{U}\, (\varphi_1 \wedge \varphi_2))$$

Explicit model checking of LTL

Employing **Büchi automata** – accepting regular languages of infinite words (ω-regular languages)

- No finite words allowed

Procedure:

1. Property formula F negated
2. Creating Büchi automaton $A_{\neg F}$ accepting exactly language of $\neg F$
3. Converting Kripke structure to Büchi automaton A_M
4. Creating product automaton $A = A_{\neg F} \times A_M$ accepting intersection of languages
5. Checking for emptiness of language accepted by A

Büchi automaton

Büchi automaton A is tuple $(\Sigma, S, S_0, \Delta, F)$:

- Σ is finite alphabet
- S is finite set of states
- $S_0 \subseteq S$ is set of initial states
- $\Delta \subseteq S \times \Sigma \times S$ is transition relation
- $F \subseteq S$ is set of accepting states

Büchi automaton similar to finite automaton, differs in accepting conditions:

- Büchi automaton A accepts infinite word ω iff there exists run of A visiting infinitely often state in F.

Generalized Büchi automaton

GBA differs from BA in definition of accepting states and accepting condition:

- *F* is *set of sets* of accepting states
- GBA accepts infite word iff there exists run visiting infitely often state from *each* set in *F*

Büchi automata – Closure properties

- Finite union: $(Q_A \cup Q_B, \Sigma, \Delta_A \cup \Delta_B, I_A \cup I_B, F_A \cup F_B)$.
  - Assuming w.l.o.g. $Q_A \cap Q_B$ empty
- Intersection: $A' = (Q', \Sigma, \Delta_1 \cup \Delta_2, I', F')$
  - $Q' = Q_A \times Q_B \times \{1, 2\}$
  - $\Delta_1 = \{((q_A, q_B, 1), a, (q'_A, q'_B, i)) | (q_A, a, q'_A) \in \Delta_A \wedge (q_B, a, q'_B) \in \Delta_B$ and if $q_A \in F_A$ then $i = 2$ else $i = 1\}$
  - $\Delta_2 = \{((q_A, q_B, 2), a, (q'_A, q'_B, i)) | (q_A, a, q'_A) \in \Delta_A \wedge (q_B, a, q'_B) \in \Delta_B$ and if $q_B \in F_B$ then $i = 1$ else $i = 2\}$
  - $I' = I_A \times I_B \times \{1\}$
  - $F' = \{(q_A, q_B, 2) | q_B \in F_B\}$

Büchi automata – Closure properties

- Concatenation $L_A.L_B$: $A' = (Q_A \cup Q_B, \Sigma, \Delta', I', F_B)$
  - $\Delta' = \Delta_A \cup \Delta_B \cup \{(q, a, q') | q' \in I_B \text{ and } \exists f \in F_A.(q, a, f) \in \Delta_A\}$
  - if $I_A \cap F_A$ is empty then $I' = I_A$ otherwise $I' = I_A \cup I_B$
- Complementation: $\forall A : \exists A'.L(A') = \Sigma^\omega \setminus L(A)$
  - Doubly exponential construction

Creating Büchi automaton for LTL formula

Property formula is negated and transformed into *negation normal form*:

- Only atomic propositions, Boolean connectives, and X, R, and U operators
- Negation at atomic propositions only

Application of rewrite rules:

- $\mathrm{F}\,p = \top\,\mathrm{U}\,p$
- $\mathrm{G}\,p = \mathit{false}\,\mathrm{R}\,p$
- $\neg(p\,\mathrm{U}\,q) = \neg p\,\mathrm{R}\,\neg q$
- $\neg(p\,\mathrm{R}\,q) = \neg p\,\mathrm{U}\,\neg q$
- $\neg\mathrm{X}\,p = \mathrm{X}\,\neg p$
- $\neg(p\,\mathrm{U}\,q) = \neg p\,\mathrm{R}\,\neg q$
- $\neg(p\,\mathrm{R}\,q) = \neg p\,\mathrm{U}\,\neg q$
- De Morgan Boolean equivalences

Creating Büchi automaton for LTL formula

Various algorithms exist – we show declarative construction

- Not optimal in terms of Büchi automata size
- Easy to describe and understand

For LTL formula f in NNF, $cl(f)$ is smallest set of formulas satisfying all following:

- $\top \in cl(f)$
- $f \in cl(f)$
- $f_1 \in cl(f) \implies \neg f_1 \in cl(f)$
- $\mathrm{X}\, f_1 \in cl(f) \implies f_1 \in cl(f)$
- $f_1 \wedge f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$
- $f_1 \vee f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$
- $f_1\, \mathrm{U}\, f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$
- $f_1\, \mathrm{R}\, f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$

$cl(f)$ is closure of sub-formulas of f under negation

Creating Büchi automaton for LTL formula

Set $M \subseteq cl(f)$ is *maximally consistent* if it satisfies following:

- $\top \in M$
- $f_1 \in M \Leftrightarrow \neg f_1 \notin M$
- $f_1 \wedge f_2 \in M \Leftrightarrow f_1 \in M \wedge f_2 \in M$
- $f_1 \vee f_2 \in M \Leftrightarrow f_1 \in M \vee f_2 \in M$

Let $cs(f)$ be set of maximally consistent subsets of $cl(f)$ – forming states of GBA

Creating Büchi automaton for LTL formula

GBA corresponding to LTL formula f is $A = (\{init\} \cup cs(f), 2^{AP}, \Delta_1 \cup \Delta 2, \{init\}, F)$:

- $(M, a, M') \in \Delta_1 \Leftrightarrow (M' \cap AP) \subseteq a \subseteq \{p \in AP. \neg p \notin M'\}$ and:
  - $\mathrm{X} f_1 \in M \Leftrightarrow f_1 \in M'$
  - $f_1 \,\mathrm{U}\, f_2 \in M \Leftrightarrow f_2 \in M \vee (f_1 \in M \wedge f_1 \,\mathrm{U}\, f_2 \in M')$
  - $f_1 \,\mathrm{R}\, f_2 \in M \Leftrightarrow f_1 \wedge f_2 \in M \vee (f_2 \in M \wedge f_1 \,\mathrm{R}\, f_2 \in M')$
- $\Delta_2 = \{(init, a, M').(M' \cap AP) \subseteq a \subseteq \{p \in AP. \neg p \notin M'\} \wedge f \in M'\}$
- $\forall f_1 \,\mathrm{U}\, f_2 \in cl(f).\{M \in cs(f).f_2 \in M \vee \neg(f_1 \,\mathrm{U}\, f_2) \in M\} \in F$

Creating Büchi automaton for LTL formula – Example

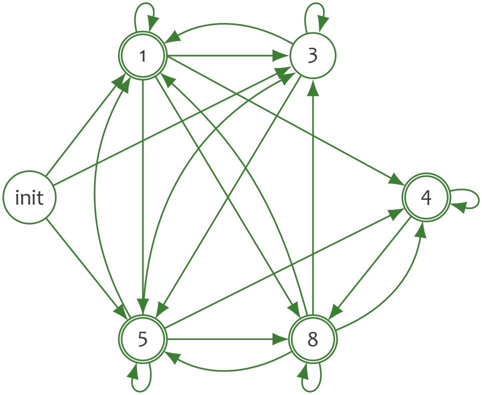
Assume LTL formula $f = p \,\mathsf{U}\, q$

$cl(f) = \{\top, p, \neg p, q, \neg q, p \,\mathsf{U}\, q\}$

$$
\begin{aligned}
cs(f) = \{\; & \\
& \{\top, p, q, p \,\mathsf{U}\, q\}_1, \\
& \{\top, p, q, \neg(p \,\mathsf{U}\, q)\}_2, \\
& \{\top, p, \neg q, p \,\mathsf{U}\, q\}_3, \\
& \{\top, p, \neg q, \neg(p \,\mathsf{U}\, q)\}_4, \\
& \{\top, \neg p, q, p \,\mathsf{U}\, q\}_5, \\
& \{\top, \neg p, q, \neg(p \,\mathsf{U}\, q)\}_6, \\
& \{\top, \neg p, \neg q, p \,\mathsf{U}\, q\}_7, \\
& \{\top, \neg p, \neg q, \neg(p \,\mathsf{U}\, q)\}_8 \}
\end{aligned}
$$

The sets in $cs(f) \cup init$ form states of GBA

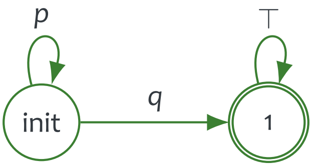
Assume LTL formula $f = p \, \mathsf{U} \, q$



Labels contain exactly atomic propositions valid in target states:

- $1 \longrightarrow 1 : p, q$
- $1 \longrightarrow 3 : p$
- $1 \longrightarrow 4 : p$
- $1 \longrightarrow 5 : q$
- $1 \longrightarrow 8 : \top$
- etc.

Simplified Büchi automaton for $f = p \,\mathsf{U}\, q$

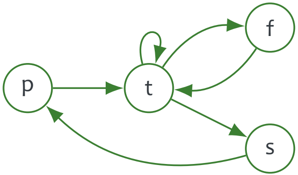


- Label p denotes set of all subsets that contain p: $\{\{p\}, \{p, q\}\}$
- Label $\top$ denotes set of all subsets of AP: $\{\{\}, \{p\}, \{q\}, \{p, q\}\}$
- Transition can be taken if there is exactly matching subset of atomic propositions.

LTL MODEL CHECKING EXAMPLE

Assume process sending data over possibly failing network

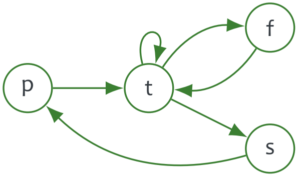
Kripke structure:



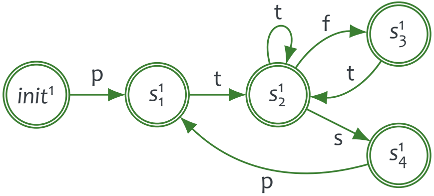
LTL MODEL CHECKING EXAMPLE

Assume process sending data over possibly failing network

Kripke structure:



Corresponding Büchi automaton:

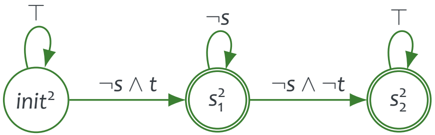


LTL MODEL CHECKING EXAMPLE

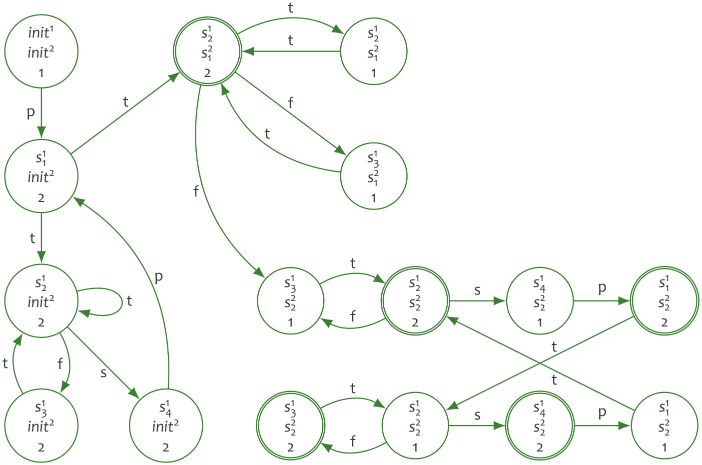
Assume property that each generated message gets delivered (is sent) eventually:

$$f : \mathsf{G}\,(t \implies (t\,\mathsf{U}\,s))$$

Büchi automaton corresponding to negated property formula $\neg f$:



Product automaton of Büchi automata – what does it say?



LTL MODEL CHECKING EXAMPLE

Conclusion: The language of the product automaton is clearly not empty (accepting runs exist), so there exists a sequence of states in the original Kripke structure satisfying the negated property formula, hence violating the original property formula.