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

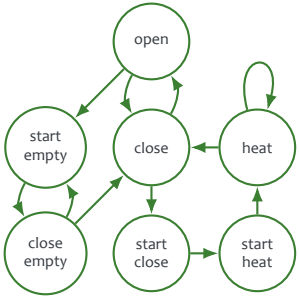
Part I: Model Checking

Model checking

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

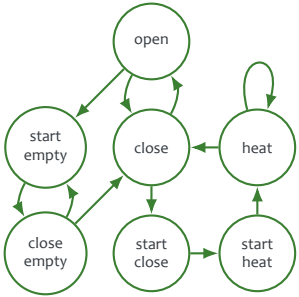
- In its basic form realized as traversal of finite graph
- Evaluates validity of property at given (initial) state
- Linear in size of graph (system model) and varying complexity in size of property (linear to exponential)
  - ...but formula size is much smaller than model

Model checking



System model

Model checking

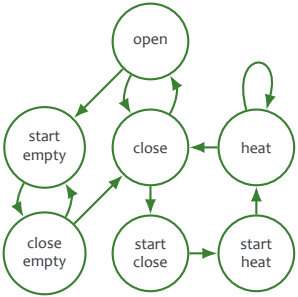


System model

G (start → F heat)

Property specification

Model checking



System model



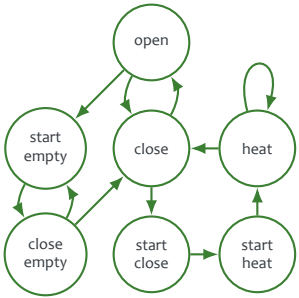


Model Checker

G (start → F heat)

Property specification

Model checking



System model

G (start → F heat)

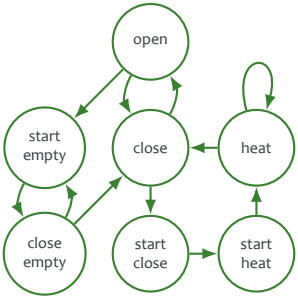
Property specification

Model Checker

Property satisfied

Property violated

Quiz

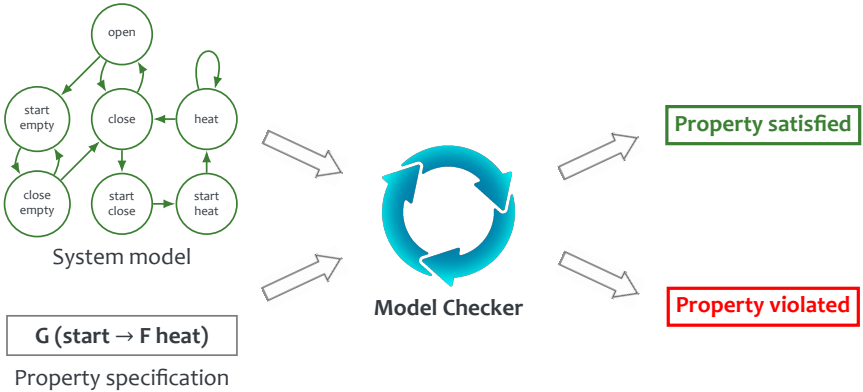


System model

G (start → F heat)

Property specification

Model checking – Practical challenges



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

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 → 10 minutes in 1990 → 0.6 seconds in 2000 → ...
- However – explosive state growth in software inherently limits scalability

MODELLING BEHAVIOUR

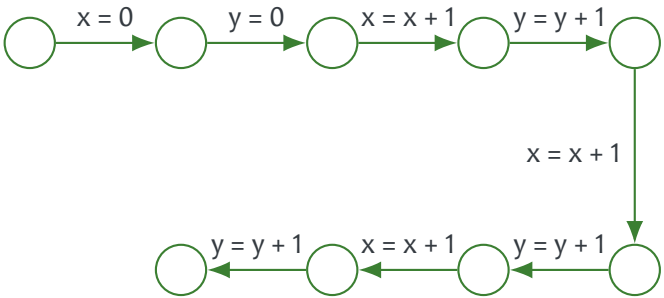
Behaviour of system (program, hardware, computer system in general) can be captured in several ways:

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

Reminder: Labelled Transition System

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

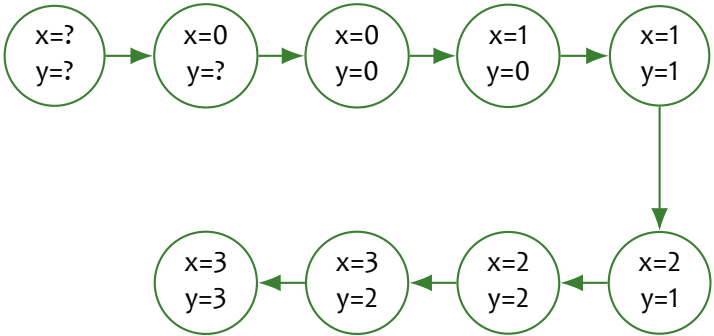
for i:=1 to 3 {
  x:=x+1;
  y:=y+1;
}
```



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 executions add self-loop to final state 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;
}
```

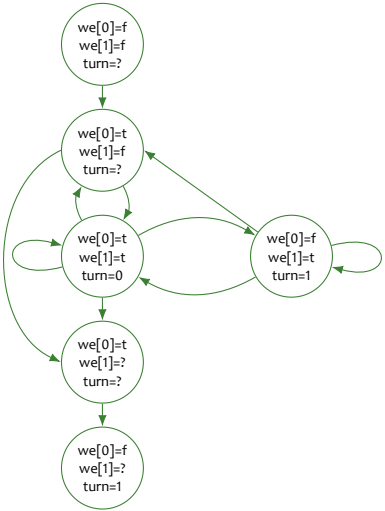
Example – Dekker's algorithm

```
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

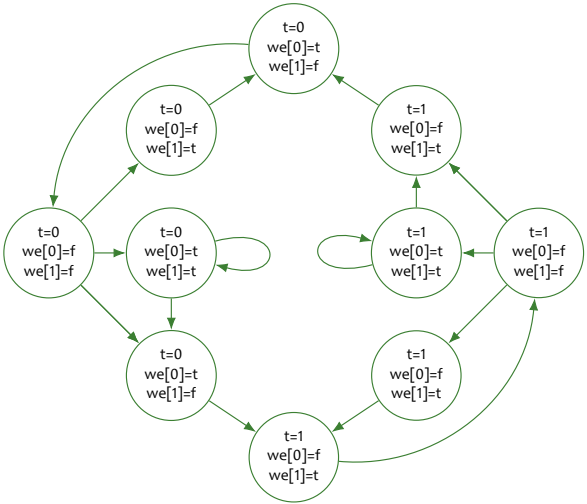
   turn = 1-id;
   wants_to_enter[id] = false;
}
```



Example – Dekker's algorithm

Composition of KS for two processes:

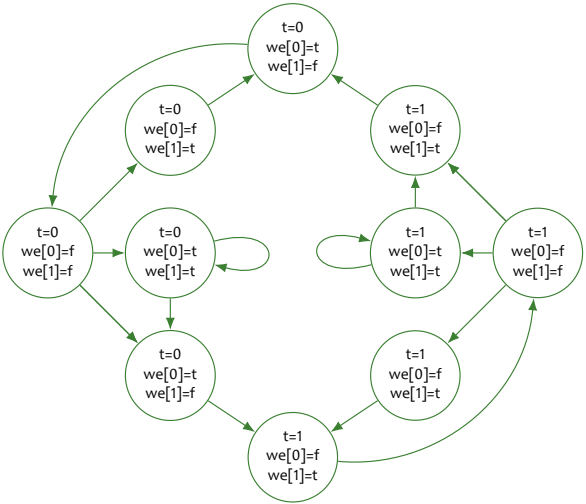
(N: idle; T: trying; C: critical; D: leaving)



Example – Dekker's algorithm

Composition of KS for two processes:
(N: idle; T: trying; C: critical; D: leaving)

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

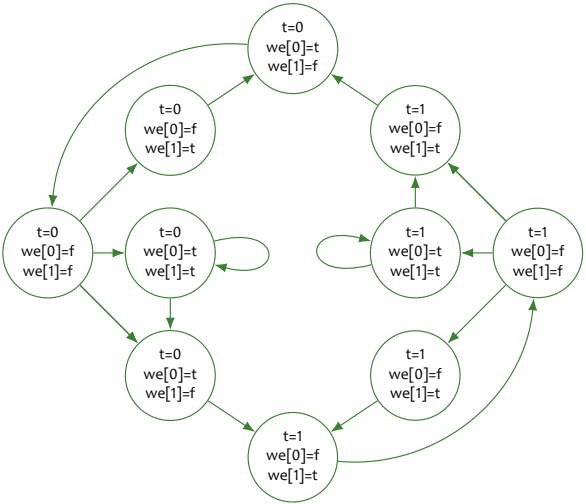


Example – Dekker's algorithm

Composition of KS for two processes:
(N: idle; T: trying; C: critical; D: leaving)

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?

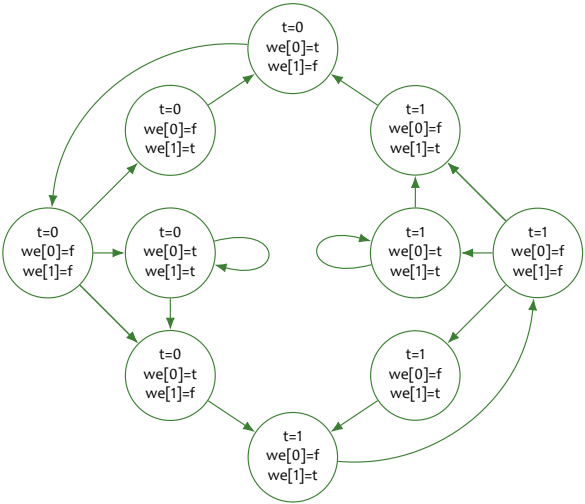


Example – Dekker's algorithm

Composition of KS for two processes:
(N: idle; T: trying; C: critical; D: leaving)

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?



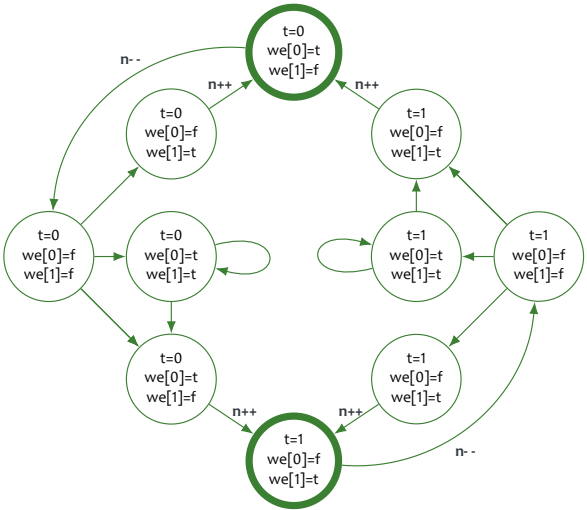
Example – Dekker's algorithm

Composition of KS for two processes:
(N: idle; T: trying; C: critical; D: leaving)

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

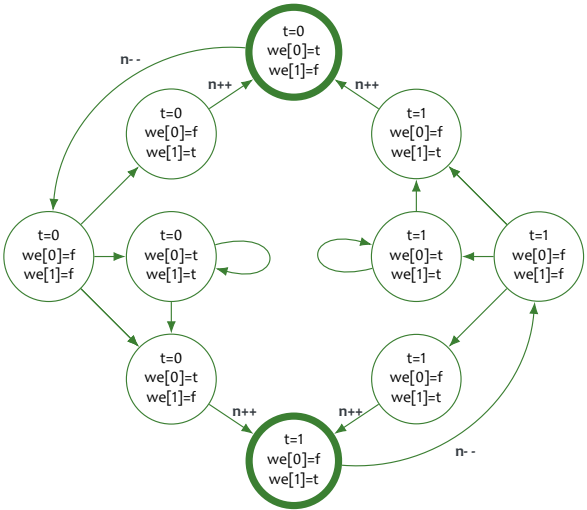
Composition of KS for two processes:
(N: idle; T: trying; C: critical; D: leaving)

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 Temporal Logic (LTL)
- Computation 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
  $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\mathsf{X}\,\varphi$, $\varphi\ \mathsf{U}\ \psi$, $\mathsf{F}\,\varphi$, $\varphi\ \mathsf{R}\ \psi$, and $\mathsf{G}\,\varphi$ 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

$$\forall i \geq 0 : (\pi_i, \pi_{i+1}) \in R.$$

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.

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

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

LTL SEMANTICS

$$
\begin{aligned}
M, \pi \vDash p &\leftrightarrow p \in L(\pi_0) \\
M, \pi \vDash \neg\varphi &\leftrightarrow \neg(M, \pi \vDash \varphi) \\
M, \pi \vDash \varphi_1 \vee \varphi_2 &\leftrightarrow M, \pi \vDash \varphi_1 \vee M, \pi \vDash \varphi_2 \\
M, \pi \vDash \varphi_1 \wedge \varphi_2 &\leftrightarrow M, \pi \vDash \varphi_1 \wedge M, \pi \vDash \varphi_2 \\
M, \pi \vDash \mathsf{X}\,\varphi &\leftrightarrow M, \pi^1 \vDash \varphi \\
M, \pi \vDash \mathsf{F}\,\varphi &\leftrightarrow \exists i \geq 0 : M, \pi^i \vDash \varphi \\
M, \pi \vDash \mathsf{G}\,\varphi &\leftrightarrow \forall i \geq 0 : M, \pi^i \vDash \varphi \\
M, \pi \vDash \varphi_1 \,\mathsf{U}\, \varphi_2 &\leftrightarrow \exists i \geq 0 : M, \pi^i \vDash \varphi_2 \wedge \forall j : 0 \leq j < i \implies M, \pi^j \vDash \varphi_1 \\
M, \pi \vDash \varphi_1 \,\mathsf{R}\, \varphi_2 &\leftrightarrow M, \pi \vDash (\mathsf{G}\,\varphi_2) \vee (\varphi_2 \,\mathsf{U}\, (\varphi_1 \wedge \varphi_2))
\end{aligned}
$$

Explicit model checking of LTL

Employing **Nondeterministic Büchi Automata** (NBA) – accepting regular languages of infinite words (ω-regular languages)

- No finite words allowed
- Stronger expressive power than deterministic BA

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

Nondeterministic Büchi automaton (NBA)

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

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

Nondeterministic 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.
- Expression powers of deterministic and nondeterministic BA differ!

Nondeterministic 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: $A' = (Q', \Sigma, \Delta_1 \cup \Delta_2, I', F')$
  - $Q' = Q_A \times Q_B \times \{1,2\}$
  - $\Delta_1 = \{((q_A, q_B, 1), a, (q'_A, q'_B, i)) \mid (q_A, a, q'_A) \in \Delta_A \wedge (q_B, a, q'_B) \in \Delta_B$ and if $q_A \in F_A$ then $i = 2$ else $i = 1\}$
  - $\Delta_2 = \{((q_A, q_B, 2), a, (q'_A, q'_B, i)) \mid (q_A, a, q'_A) \in \Delta_A \wedge (q_B, a, q'_B) \in \Delta_B$ and if $q_B \in F_B$ then $i = 1$ else $i = 2\}$
  - $I' = I_A \times I_B \times \{1\}$
  - $F' = \{(q_A, q_B, 2) \mid q_B \in F_B\}$

Nondeterministic Büchi automata – Closure properties

- Concatenation of NFA A and NBA B: $L_A.L_B$: $A' = (Q_A \cup Q_B, \Sigma, \Delta', I', F_B)$
  - $\Delta' = \Delta_A \cup \Delta_B \cup \{(q, a, q') \mid q' \in I_B \text{ and } \exists f \in F_A.(q, a, f) \in \Delta_A\}$
  - if $I_A \cap F_A$ is empty then $I' = I_A$ otherwise $I' = I_A \cup I_B$

- Complementation: $\forall A \exists A' : L(A') = \Sigma^\omega \backslash L(A)$
  - Exponential construction: $2^{O(n \log n)}$

Generalized Nondeterministic Büchi automaton (GNBA)

Generalized Nondeterministic Büchi Automaton differs from NBA in definition of accepting states and accepting condition:

- F is *set of sets* of accepting states
- *GNBA* accepts infinite word iff there exists run visiting infinitely often state from *each* set in F
- GNBA can be converted to NBA by employing similar procedure used in NBA intersection (a copy of NBA for each set of accepting states)

Creating Büchi automaton for LTL formula

Property formula φ is transformed into *normal form*:

- Only T, atomic propositions, $\neg$, $\wedge$, and X and U operators
- Assume $\varphi \neq \mathsf{T}$, otherwise trivial

Application of rewrite rules:

- $\mathsf{F}\, p = \mathsf{T}\, \mathsf{U}\, p$
- $\mathsf{G}\, p = \neg F \neg p$
- $p\, \mathsf{R}\, q = \neg(\neg p\, \mathsf{U}\, \neg q)$
- $\neg \mathsf{X}\, p = \mathsf{X}\, \neg p$
- De Morgan Boolean equivalences

Various algorithms exist – we show declarative construction

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

CLOSURE OF φ

For LTL formula f in normal form, $closure(\varphi)$ contains all its subformulae and their negations

E.g.: for $\varphi = a \,\mathrm{U}\, (\neg a \wedge b)$, $closure(\varphi) = \{a, b, \neg a, \neg b, \neg a \wedge b, \neg(\neg a \wedge b), \varphi, \neg\varphi\}$

$closure(\varphi)$ is closure of sub-formulae of φ under negation

- We identify $\neg\neg\psi$ with ψ

Elementary set

- Set of formulae $B \subseteq closure(\varphi)$ is called *elementary* if B is the set of all formulae $\psi \in closure(\varphi)$ satisfying:
  - It cannot contain propositional contradiction
  - Must be *locally consistent* with respect to U operator
  - Must be *maximal*

Elementary set conditions

1. B is *consistent* with respect to propositional logic, i.e., for all $\varphi_1 \wedge \varphi_2, \psi \in closure(\varphi)$:
   - $\varphi_1 \wedge \varphi_2 \in B \iff \varphi_1 \in B$ and $\varphi_2 \in B$
   - $\psi \in B \implies \neg\psi \notin B$
   - $\top \in closure(\varphi) \implies \top \in B$
2. B is *locally consistent* with respect to the until operator, i.e., for all $\varphi_1 \mathrel{U} \varphi_2 \in closure(\varphi)$:
   - $\varphi_2 \in B \implies \varphi_1 \mathrel{U} \varphi_2 \in B$
   - $\varphi_1 \mathrel{U} \varphi_2 \in B$ and $\varphi_2 \notin B \implies \varphi_1 \in B$
3. B is *maximal*, i.e., for all $\psi \in closure(\varphi)$:
   - $\psi \notin B \implies \neg\psi \in B$

Elementary set – Examples

- Let $\varphi = a \,\mathsf{U}\, (\neg a \wedge b)$
- Set $B = \{a, b, \varphi\}$ is consistent, locally consistent, but not maximal.
  - For $\neg a \wedge b \in closure(\varphi)$: $\neg a \wedge b \notin B$ and $\neg(\neg a \wedge b) \notin B$.

Elementary set – Examples

- Let $\varphi = a \,\mathsf{U}\, (\neg a \wedge b)$
- Set $B = \{a, b, \varphi\}$ is consistent, locally consistent, but not maximal.
  - For $\neg a \wedge b \in closure(\varphi)$: $\neg a \wedge b \notin B$ and $\neg(\neg a \wedge b) \notin B$.
- Set $C = \{a, b, \neg a \wedge b, \varphi\}$ is propositionally inconsistent.

Elementary set – Examples

- Let $\varphi = a \,\mathsf{U}\, (\neg a \wedge b)$
- Set $B = \{a, b, \varphi\}$ is consistent, locally consistent, but not maximal.
  - For $\neg a \wedge b \in closure(\varphi)$: $\neg a \wedge b \notin B$ and $\neg(\neg a \wedge b) \notin B$.
- Set $C = \{a, b, \neg a \wedge b, \varphi\}$ is propositionally inconsistent.
- Set $D = \{\neg a, \neg b, \neg(\neg a \wedge b), \varphi\}$ is propositionally consistent but locally inconsistent w.r.t. until operator.
  - $a \,\mathsf{U}\, (\neg a \wedge b) \in D$ and $\neg a \wedge b \notin D$, but $a \notin D$.
  - There is no path π such that $\pi \vDash \neg a$, $\pi \vDash \neg(\neg a \wedge b)$, and $\pi \vDash a \,\mathsf{U}\, (\neg a \wedge b)$.

Elementary set – Examples

- Let $\varphi = a \,\mathsf{U}\, (\neg a \wedge b)$
- Set $B = \{a, b, \varphi\}$ is consistent, locally consistent, but not maximal.
  - For $\neg a \wedge b \in closure(\varphi)$: $\neg a \wedge b \notin B$ and $\neg(\neg a \wedge b) \notin B$.
- Set $C = \{a, b, \neg a \wedge b, \varphi\}$ is propositionally inconsistent.
- Set $D = \{\neg a, \neg b, \neg(\neg a \wedge b), \varphi\}$ is propositionally consistent but locally inconsistent w.r.t. until operator.
  - $a \,\mathsf{U}\, (\neg a \wedge b) \in D$ and $\neg a \wedge b \notin D$, but $a \notin D$.
  - There is no path π such that $\pi \vDash \neg a$, $\pi \vDash \neg(\neg a \wedge b)$, and $\pi \vDash a \,\mathsf{U}\, (\neg a \wedge b)$.

Elementary sets:

$B_1 = \{a, b, \neg(\neg a \wedge b), \varphi\}$
$B_2 = \{a, b, \neg(\neg a \wedge b), \neg\varphi\}$
$B_3 = \{a, \neg b, \neg(\neg a \wedge b), \varphi\}$
$B_4 = \{a, \neg b, \neg(\neg a \wedge b), \neg\varphi\}$
$B_5 = \{\neg a, \neg b, \neg(\neg a \wedge b), \neg\varphi\}$
$B_6 = \{\neg a, b, \neg a \wedge b, \varphi\}$

GNBA $\mathcal{G}_\varphi$ FOR LTL φ

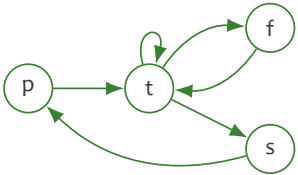
Let φ be LTL formula over AP. Then $\mathcal{G}_\varphi = (Q, 2^{AP}, \Delta, Q_0, F)$, where:

- Q is set of all elementary sets of $B \subseteq closure(\varphi)$
- AP is set of atomic propositions
- $Q_0 = \{B \in Q \mid \varphi \in B\}$
- $F = \{F_{\varphi_1 \cup \varphi_2} \mid \varphi_1 \cup \varphi_2 \in closure(\varphi)\}$ where
$$F_{\varphi_1 \cup \varphi_2} = \{B \in Q \mid \varphi_1 \cup \varphi_2 \notin B \text{ or } \varphi_2 \in B\}$$
- Δ : transition (B, A, B') is defined iff:
  - $A = B \cap AP$, and
  - for every $\mathrm{X}\psi \in closure(\varphi)$: $\mathrm{X}\psi \in B \Leftrightarrow \psi \in B'$, and
  - for every $\varphi_1 \cup \varphi_2 \in closure(\varphi)$:
$$\varphi_1 \cup \varphi_2 \in B \Leftrightarrow (\varphi_2 \in B \vee (\varphi_1 \in B \wedge \varphi_1 \cup \varphi_2 \in B')).$$

LTL MODEL CHECKING EXAMPLE

Assume process sending data over possibly failing network

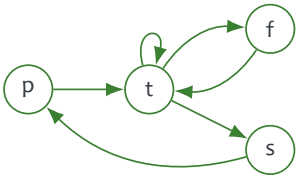
Kripke structure:



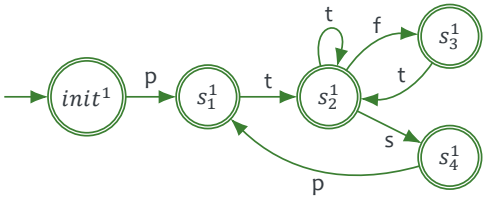
LTL MODEL CHECKING EXAMPLE

Assume process sending data over possibly failing network

Kripke structure:



Corresponding Büchi automaton:

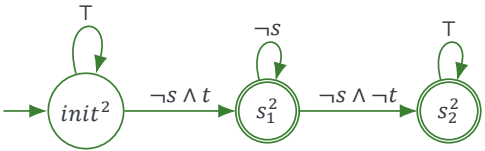


LTL MODEL CHECKING EXAMPLE

Assume property that each generated message remains transmitted until sent:

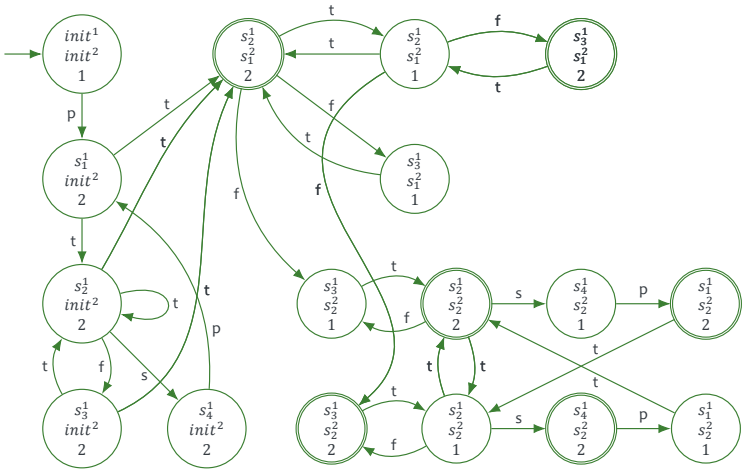
$$f : \mathsf{G}(t \implies (t \,\mathsf{U}\, s))$$

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



LTL MODEL CHECKING EXAMPLE

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.

Summary

- **Model checking** systematically traverses system state space
- Requires formal system model (Kripke structure) and property specification (in temporal logic)
- **LTL model checking** based on checking language emptiness of product Büchi automata
  - Büchi automaton for negated specification formula representing violation runs
  - Büchi automaton for system model (Kripke structure)
  - Product automaton of the two above accepting specification violations of system model