NSWI101: SYSTEM BEHAVIOUR MODELS AND VERIFICATION

2. MODEL CHECKING

Jan Kofroň
- Model checking
- Linear Temporal Logic (LTL)
- Büchi automata
Part I: Model Checking
Model checking is a process of determining whether given model $M$ satisfies given property $\varphi$ ($M \models \varphi$).

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

System model

open

start empty

close empty

close

heat

start close

start heat

AG (start → AF heat)

Property specification

Model Checker

Property satisfied

Property violated
**Model Checking**

System model

**AG (start → AF heat)**

Property specification
MODEL CHECKING

System model

AG (start → AF heat)

Model Checker

Property specification
MODEL CHECKING

System model

AG (start → AF heat)

Property specification

Property satisfied

Property violated
MODEL CHECKING – PRACTICAL CHALLENGES

System model

AG (start → AF heat)
Property specification

Model construction
Property expression

State space explosion
Error trace interpretation

Property satisfied
Property violated
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

\[
\begin{align*}
x &:= 0; \\
y &:= 0; \\
\text{for } i := 1 \text{ to } 3 \{ \\
&\quad x := x + 1; \\
&\quad y := y + 1; \\
\} 
\end{align*}
\]
x:=0;
y:=0;

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

\[ x=0, y=0 \]
\[ x=1, y=0 \]
\[ x=1, y=1 \]
\[ x=2, y=2 \]
\[ x=2, y=1 \]
\[ x=3, y=2 \]
\[ x=3, y=3 \]
**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)
- $L : S \rightarrow 2^{AP}$ – 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 \((s \rightarrow t) \in R\) 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.
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**

```c
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;
}
```
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?

Artificial model variable can help.
Correctness property: "n is always less than 2."
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?
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.
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
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 defined inductively, similarly to propositional logic:

Let $\text{AP}$ be a finite set of Boolean variables (atomic propositions). The set of LTL formulae over $\text{AP}$ is defined as:

- If $p \in \text{AP}$ then $p$ is LTL formula.
- If $\varphi$ and $\psi$ are LTL formulae then $\neg \varphi$, $\varphi \lor \psi$, $\varphi \land \psi$, $X \varphi$, $\varphi U \psi$, $F \varphi$, $\varphi R \psi$, and $G \varphi$ 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 $\forall i > 0. (\pi_i, \pi_{i+1}) \in R$

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 $i \geq 0$, $\pi^i$ stands for i-th suffix of $\pi$: $\pi^i = \pi_i, \pi_{i+1}, \pi_{i+2}, \ldots$

Let $M$ be Kripke structure, $\varphi$ be LTL formula, $\pi$ be path in $M$ and $s$ be state of $M$.

$M, \pi \models \varphi$: Path $\pi$ from $M$ satisfies $\varphi$

$M, s \models \varphi$: State $s$ from $M$ satisfies $\varphi$

- $M, s \models \varphi \iff \forall \pi. \pi_0 = s : M, \pi \models \varphi$
LTL SEMANTICS

\[ M, \pi \models p \iff p \in L(\pi_0) \]
\[ M, \pi \models \neg \varphi \iff \neg (M, \pi \models \varphi) \]
\[ M, \pi \models \varphi_1 \lor \varphi_2 \iff M, \pi \models \varphi_1 \lor M, \pi \models \varphi_2 \]
\[ M, \pi \models \varphi_1 \land \varphi_2 \iff M, \pi \models \varphi_1 \land M, \pi \models \varphi_2 \]
\[ M, \pi \models X \varphi \iff M, \pi^1 \models \varphi \]
\[ M, \pi \models F \varphi \iff \exists i \geq 0. M, \pi^i \models \varphi \]
\[ M, \pi \models G \varphi \iff \forall i \geq 0. M, \pi^i \models \varphi \]
\[ M, \pi \models \varphi_1 U \varphi_2 \iff \exists i \geq 0. M, \pi^i \models \varphi_2 \land \forall j. 0 \leq j < i \implies M, \pi^j \models \varphi_1 \]
\[ M, \pi \models \varphi_1 R \varphi_2 \iff (G \varphi_2) \lor (\varphi_2 U (\varphi_1 \land \varphi_2)) \]
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 $A$ is tuple $(\Sigma, S, S_0, \Delta, F)$:

- $\Sigma$ 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 $\omega$ 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 infinite word iff there exists run visiting infinitely often state from each set in $F$
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: \(A' = (Q', \Sigma, \Delta_1 \cup \Delta_2, I', F')\)

\(Q' = Q_A \times Q_B \times \{1, 2\}\)

\(\Delta_1 = \{((q_A, q_B, 1), a, (q'_A, q'_B, i)) | (q_A, a, q'_A) \in \Delta_A \land (q_B, a, q'_B) \in \Delta_B \land\)

\text{if } q_A \in F_A \text{ then } i = 2 \text{ else } i = 1\}\)

\(\Delta_2 = \{((q_A, q_B, 2), a, (q'_A, q'_B, i)) | (q_A, a, q'_A) \in \Delta_A \land (q_B, a, q'_B) \in \Delta_B \land\)

\text{if } q_B \in F_B \text{ then } i = 1 \text{ 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:
- $F p = \top U p$
- $G p = false R p$
- $\neg(p U q) = \neg p R \neg q$
- $\neg(p R q) = \neg p U \neg q$
- $\neg X p = X \neg p$
- $\neg(p U q) = \neg p R \neg q$
- $\neg(p R q) = \neg p U \neg q$
- 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:

- $\top \in cl(f)$
- $f \in cl(f)$
- $f_1 \in cl(f) \implies \neg f_1 \in cl(f)$
- $Xf_1 \in cl(f) \implies f_1 \in cl(f)$
- $f_1 \land f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$
- $f_1 \lor f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$
- $f_1 \mathbin{U} f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$
- $f_1 \mathbin{R} f_2 \in cl(f) \implies f_1, f_2 \in cl(f)$

$cl(f)$ is closure of sub-formulas of $f$ under negation
CREATING BÜCHI AUTOMATON FOR LTL FORMULA

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

- $\top \in M$
- $f_1 \in M \iff \neg f_1 \notin M$
- $f_1 \land f_2 \in M \iff f_1 \in M \land f_2 \in M$
- $f_1 \lor f_2 \in M \iff f_1 \in M \lor f_2 \in M$

Let $\text{cs}(f)$ be set of maximally consistent subsets of $\text{cl}(f)$ – forming states of GBA
Creating Büchi Automaton for LTL Formula

GBA corresponding to LTL formula \( f \) is

\[ A = (\{ \text{init} \} \cup \text{cs}(f), \mathcal{Z}^{AP}, \Delta_1 \cup \Delta_2, \{ \text{init} \}, F) : \]

- \((M, a, M') \in \Delta_1 \iff (M' \cap AP) \subseteq a \subseteq \{ p \in AP. \neg p \notin M' \} \) and:
  - \( Xf_1 \in M \iff f_1 \in M' \)
  - \( f_1 U f_2 \in M \iff f_2 \in M \lor (f_1 \in M \land f_1 U f_2 \in M') \)
  - \( f_1 R f_2 \in M \iff f_1 \land f_2 \in M \lor (f_2 \in M \land f_1 R f_2 \in M') \)

- \( \Delta_2 = \{(\text{init}, a, M').(M' \cap AP) \subseteq a \subseteq \{ p \in AP. \neg p \notin M' \} \land f \in M' \} \)

- \( \forall f_1 U f_2 \in \text{cl}(f).\{ M \in \text{cs}(f).f_2 \in M \lor \neg(f_1 U f_2) \in M \} \in F \)
CREATING BÜCHI AUTOMATON FOR LTL FORMULA – EXAMPLE

Assume LTL formula \( f = p \lor q \)

\[ cl(f) = \{ \top, p, \neg p, q, \neg q, p \lor q \} \]

\[ cs(f) = \{ \]
\[ \{ \top, p, q, p \lor q \}_1, \]
\[ \{ \top, p, q, \neg(p \lor q) \}_2, \]
\[ \{ \top, p, \neg q, p \lor q \}_3, \]
\[ \{ \top, p, \neg q, \neg(p \lor q) \}_4, \]
\[ \{ \top, \neg p, q, p \lor q \}_5, \]
\[ \{ \top, \neg p, q, \neg(p \lor q) \}_6, \]
\[ \{ \top, \neg p, \neg q, p \lor q \}_7, \]
\[ \{ \top, \neg p, \neg q, \neg(p \lor q) \}_8 \} \]

The sets in \( cs(f) \cup \text{init} \) form states of GBA
Assume LTL formula $f = p U q$

Labels contain exactly atomic propositions valid in target states:
- $1 \rightarrow 1 : p, q$
- $1 \rightarrow 3 : p$
- $1 \rightarrow 4 : p$
- $1 \rightarrow 5 : q$
- $1 \rightarrow 8 : \top$
- etc.
Simplified Büchi automaton for $f = p \cup q$

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:
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:

\[ f : \Box (t \implies (t U s)) \]

Büchi automaton corresponding to negated property formula \( \neg f \):

![Automaton Diagram]
Product automaton of Büchi automata – what does it say?
**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.