IA169 System Verification and Assurance

LTL Model Checking

Jiří Barnat

Motivation

Checking Quality

- Testing is incomplete, gives no guarantees of correctness.
- Deductive verification is expensive.

Typical reasons for system failure

- Unexpected or boundary input values.
- Interaction of system components.
- Parallelism (difficult to test).

Model Checking

- Automated verification process for ...
- ... parallel and distributed systems.

Verification of Parallel and Reactive Programs

Parallel Programs

Parallel Composition

- Components concurrently contribute to the transformation of a computation state.
- The meaning comes from interleaving of actions (transformation steps) of individual components.

Meaning Functions Do Not Compose

- Meaning function of a composition cannot be obtain as composition of meaning functions of participating components.
- The result depends on particular interleaving.

Example of Incomposability

Parallel System

- System: (y=x; y++; x=y) ∥ (y=x; y++; x=y)
- Input-output variable x
- Meaning function of both processes is λ`x->x+1`.
- The composition is: `(`λ`x->x+1)`$\cdot$`(`λ`x->x+1)`.
- `(`λ`x->x+1)`$\cdot$`(`λ`x->x+1)` 0 = 2

Two Different System Runs

- State $= (x, y_1, y_2)$
- $(0,\text{-},\text{-}) \xrightarrow{y_1=x} (0,0,\text{-}) \xrightarrow{y_2=x} (0,0,0) \xrightarrow{y_1++} \xrightarrow{x=y_1} (1,1,0) \xrightarrow{y_2++} \xrightarrow{x=y_2} (\mathbf{1},1,1)$
- $(0,\text{-},\text{-}) \xrightarrow{y_1=x} (0,0,\text{-}) \xrightarrow{y_1++} \xrightarrow{x=y_1} (1,1,\text{-}) \xrightarrow{y_2=x} (1,1,1) \xrightarrow{y_2++} \xrightarrow{x=y_2} (\mathbf{2},1,2)$

Properties of Parallel Programs

Observation

- Specific timing of events related to interaction of components is a form of (part of) input.
- Asynchronous parallel system can be viewed as reactive as there are unknown inputs at the time of execution.

Consequence

- For parallel and reactive systems it is difficult to specify the intended behaviour using pre- and post-conditions.

Properties of Parallel/Reactive Programs

Examples of Specification

- Events A and B happens before event C.
- User is not allowed to enter a new value until the system processes the previous one.
- Procedure X cannot be executed simultaneously by processes P and Q (mutual exclusion).
- Every action A is immediately followed by a sequence of actions B,C and D.

Turning into Formal Language

- Use of Modal and Temporal Logics.
- Amir Pnueli, 1977

Deductive Verification for Modal and Temporal Logic

Observation

- Systems similar to Hoare Logic may be built for modal and temporal logic.
- Even more demanding on personal.
- For parallel and reactive systems exhibits similar disadvantages as techniques built on top of pre- and post-conditions.

Model checking

- Alternative way of formal verification of systems.
- Specification given with formulae of some temporal logic.
- Based on state-space exploration.

Model Checking

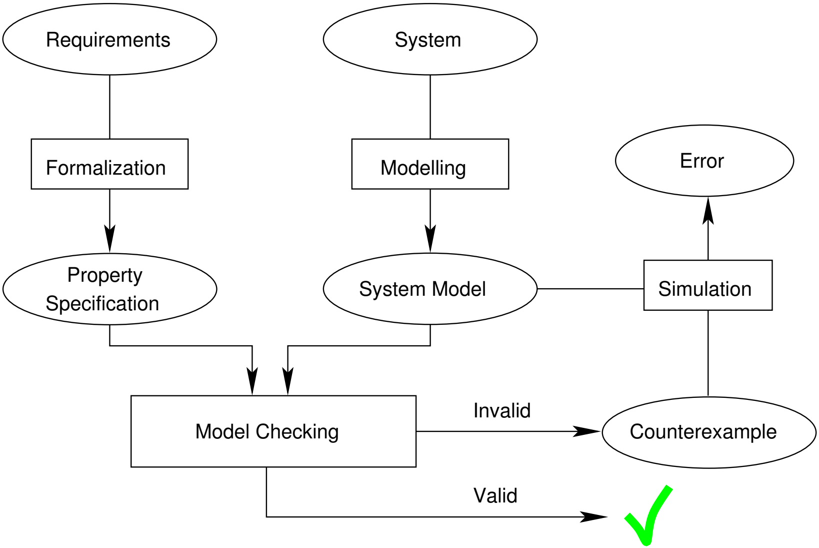
Model Checking – Overview

- Build a formal model $\mathcal{M}$ of the system under verification.
- Express specification as a formula φ of selected temporal logic.
- Decide, if $\mathcal{M} \models \varphi$. That is, if $\mathcal{M}$ is a model of formula φ. (Hence the name.)

Optionally

- As a side effect of the decision a **counterexample** may be produced.
- The counterexample is a sequence of states witnessing violation (in the case the system is erroneous) of the formula.
- **Model checking (the decision process) can be fully automated for all finite (and some infinite) models of systems.**

Model Checking – Schema



Automated Tools for Model Checking

Model Checkers

- Software tools that can decide validity of a formula over a model of system under verification.
- SPIN, UppAal, SMV, Prism, DIVINE ...

Modelling Languages

- Processes described as extended finite state machines.
- Extension allows to use shared or local variables and guard execution of a transition with a Boolean expression.
- Optionally, some transitions may be synchronised with transitions of other finite state machines/processes.

Section

Modelling and Formalisation of Verified Systems

Reminder

- System can be viewed as a set of states that are walked along by executing instructions of the program.
- State = valuation of modelled variables.

Atomic Propositions

- Basic statements describing qualities of individual states, for example: $max(x, y) \geq 3$.
- Validity of atomic proposition for a given state must be decidable with information merely encoded by the state.
- Amount of observable events and facts depends on amount of abstraction used during the system modelling.

Kripke Structure

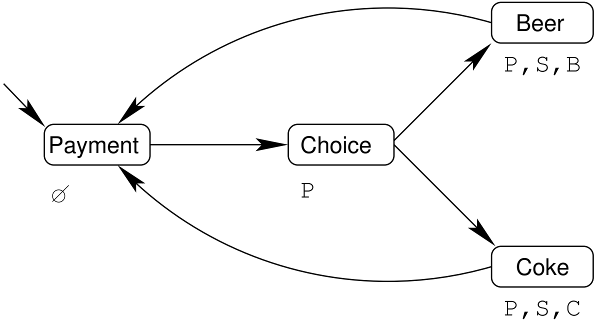
- Let AP be a set of atomic propositions.
- Kripke structure is a quadruple (S, T, I, s_0), where
  - S is a (finite) set of states,
  - $T \subseteq S \times S$ is a transition relation,
  - $I : S \rightarrow 2^{AP}$ is an interpretation of AP.
  - $s_0 \in S$ is an initial state.

Kripke Transition System

- Let Act be a set of instructions executable by the program.
- Kripke structure can be extended with transition labelling to form a Kripke Transitions System.
- Kripke Transition System is a five-tuple $(S, T, I, s_0, \mathcal{L})$, where
  - (S, T, I, s_0) is Kripke Structure,
  - $\mathcal{L} : T \rightarrow Act$ is labelling function.

Kripke Structure – Example

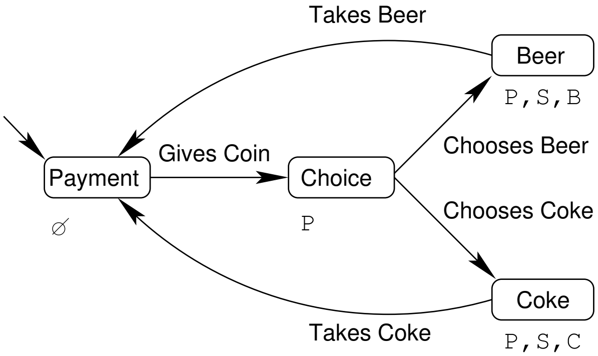
Kripke Structure



AP={P – Paid, S – Served, C – Coke, B – Beer}

Kripke Structure – Example

Kripke Transition System



AP={P – Paid, S – Served, C – Coke, B – Beer}

System Run

Run

- Maximal path (such that it cannot be extended) in the graph induced by Kripke Structure starting at the initial state.
- Let $M = (S, T, I, s_0)$ be a Kripke structure. Run is a sequence of states $\pi = s_0, s_1, s_2, \ldots$ such that $\forall i \in \mathbb{N}_0.(s_i, s_{i+1}) \in T$.

Finite Paths and Runs

- Some finite path $\pi = s_0, s_1, s_2, \ldots, s_k$ cannot be extended if $\nexists s_{k+1} \in S.(s_k, s_{k+1}) \in T$.
- Technically, we will turn maximal finite path into infinite by repeating the very last state.
- Maximal path $s_0, \ldots, s_k$ will be understood as infinite run $s_0, \ldots, s_k, s_k, s_k, \ldots$.

Implicit and Explicit System Description

Observation

- Usually, Kripke structure that captures system behaviour is not given by full enumeration of states and transitions (explicitly), but it is given by the program source code (implicitly).
- Implicit description tends to be exponentially more succinct.

State-Space Generation

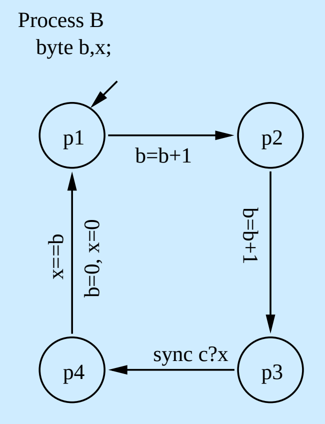
- Computation of explicit representation from the implicit one.
- Interpretation of implicit representation must be formally precise.

Practise

- Programming languages do not have precise formal semantics.
- Model checkers often build on top of modelling languages.

En Example of Modelling Language – DVE

- Finite Automaton
  - States (Locations)
  - Initial state
  - Transitions
  - (Accepting states)
- Transitions Extended with
  - Guards
  - Synchronisation and Value Passing
  - Effect (Assignment)
- Local Variables
  - `integer`, `byte`
  - `channel`

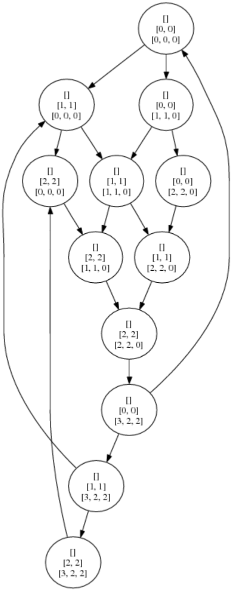


```
channel {byte} c[0];

process A {
byte a;
state q1,q2,q3;
init q1;
trans
q1→q2 { effect a=a+1; },
q2→q3 { effect a=a+1; },
q3→q1 { sync c!a; effect a=0; };
}

process B {
byte b,x;
state p1,p2,p3,p4;
init p1;
trans
p1→p2 { effect b=b+1; },
p2→p3 { effect b=b+1; },
p3→p4 { sync c?x; },
p4→p1 { guard x==b; effect b=0, x=0; };
}

system async;
```



Semantics Shown By Interpretation

State: []; A:[q1, a:0]; B:[p1, b:0, x:0]
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }
1 ⟨1.0⟩: p1 → p2 { effect b = b+1; }
Command:1

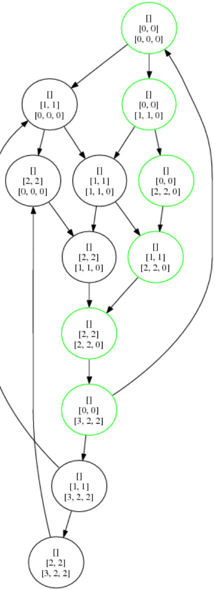
State: []; A:[q1, a:0]; B:[p2, b:1, x:0]
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }
1 ⟨1.1⟩: p2 → p3 { effect b = b+1; }
Command:1

State: []; A:[q1, a:0]; B:[p3, b:2, x:0]
0 ⟨0.0⟩: q1 → q2 { effect a = a+1; }
Command:0

State: []; A:[q2, a:1]; B:[p3, b:2, x:0]
0 ⟨0.1⟩: q2 → q3 { effect a = a+1; }
Command:0

State: []; A:[q3, a:2]; B:[p3, b:2, x:0]
0 ⟨0.2&1.2⟩: q3 → q1 { sync c!a; effect a = 0; }
p3 → p4 { sync c?x; }
Command:0

State: []; A:[q1, a:0]; B:[p4, b:2, x:2]



Formalising System Properties

Specification as Languages of Infinite Words

Problem

- How to formally describe properties of a single run?
- How to mechanically check for their satisfaction?

Solution

- Employ finite automaton as a mechanical observer of run.
- Runs are infinite.
- Finite automata for infinite words (ω-regular languages).
- Büchi acceptance condition – automaton accepts a word if it passes through an accepting state infinitely many often.

Büchi automata

- Büchi automaton is a tuple $A = (\Sigma, S, s, \delta, F)$, where
  - Σ is a finite set of symbols,
  - S is a finite set f states,
  - $s \in S$ is an initial state,
  - $\delta : S \times \Sigma \rightarrow 2^S$ is transition relation, and
  - $F \subseteq S$ is a set of accepting states.

Language accepted by a Büchi automaton

- Run ρ of automaton A over infinite word $w = a_1 a_2 \ldots$ is a sequence of states $\rho = s_0, s_1, \ldots$ such that $s_0 \equiv s$ and $\forall i : s_i \in \delta(s_{i-1}, a_i)$.
- $inf(\rho)$ – Set of states that appear infinitely many time in ρ.
- Run ρ is accepting if and only if $inf(\rho) \cap F \neq \emptyset$.
- Language accepted with an automaton A is a set of all words for which an accepting run exists. Denoted as $L(A)$.

Shortcuts in Transition Guards

Observation

- Let AP={X,Y,Z}.
- Transition labelled with $\{X\}$ denotes that X must hold true upon execution of the transition, while Y and Z are false.
- If we want to express that X is true, Z is false, and for Y we do not care, we have to create two transitions labelled with $\{X\}$ and $\{X, Y\}$.

APs as Boolean Formulae

- Transitions between the two same states may be combined and labelled with a Boolean formula over atomic propositions.

Example

- Transitions {X}, {Y}, {X,Y}, {X,Z}, {Y,Z} a {X,Y,Z} can be combined into a single one labelled with $X \vee Y$.
- If there are no restrictions upon execution of the transition, it may be labelled with $\mathit{true} \equiv X \vee \neg X$.

Task: Express with a Büchi automaton

System

- Vending machine as seen before.
- $\Sigma = 2^{\{P,S,C,B\}}$,
- $Paid = \{A \in \Sigma \mid P \in A\}$, $Served = \{A \in \Sigma \mid S \in A\}$, ...

Express the following properties

- Vending machine serves at least one drink.
- Vending machine serves at least one coke.

- Vending machine serves infinitely many drinks.
- Vending machine serves infinitely many beers.

- Vending machine does not serve a drink without being paid.
- After being paid, vending machine always serve a drink.

Linear Temporal Logic

Linear Temporal Logic (LTL) Informally

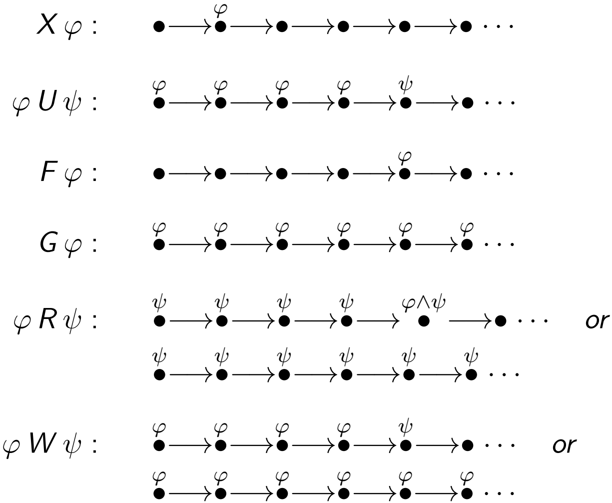
Formula φ

- Is evaluated on top of a single run of Kripke structure.
- Express validity of APs in the states along the given run.

Temporal Operators of LTL

- $F\,\varphi$ — φ holds true eventually (Future).
- $G\,\varphi$ — φ holds true all the time (Globally).
- $\varphi\,U\,\psi$ — φ holds true until eventually ψ holds true (Until).
- $X\,\varphi$ — φ is valid after execution of one transition (Next).
- $\varphi\,R\,\psi$ — ψ holds true until $\varphi \wedge \psi$ holds true (Release).
- $\varphi\,W\,\psi$ — until, but ψ may never become true (Weak Until).

Graphical Representation of LTL Temporal Operators



Syntax of LTL

Let AP be a set of atomic propositions.

- If $p \in AP$, then p is an LTL formula.
- If φ is an LTL formula, then $\neg\varphi$ is an LTL formula.
- If φ and ψ are LTL formulae, then $\varphi \vee \psi$ is an LTL formula.
- If φ is an LTL formula, then $X\,\varphi$ is an LTL formula.
- If φ and ψ are LTL formulae, then $\varphi\,U\,\psi$ is an LTL formula.

Alternatively

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\,\varphi \mid \varphi\,U\,\varphi$$

Syntactic shortcuts

Propositional Logic

- $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$
- $\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi$
- $\varphi \Leftrightarrow \psi \equiv (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

Temporal operators

- $F\,\varphi \equiv \mathit{true}\; U\,\varphi$
- $G\,\varphi \equiv \neg F \neg\varphi$
- $\varphi\, R\, \psi \equiv \neg(\neg\varphi\; U \neg\psi)$
- $\varphi\, W\, \psi \equiv \varphi\, U\, \psi \vee G\,\varphi$

Alternative syntax

- $F\varphi \equiv \Diamond\varphi$
- $G\varphi \equiv \Box\varphi$
- $X\varphi \equiv \circ\varphi$

Models of LTL Formulae

Model of an LTL formula

- Let AP be a set of atomic propositions.
- Model of an LTL formula is a run π of Kripke structure.

Notation

- Let $\pi = s_0, s_1, s_2, \ldots$.
- Suffix of run π starting at s_k is denoted as $\pi^k = s_k, s_{k+1}, s_{k+2}, \ldots$.
- K-th state of the run, is referred to as $\pi(k) = s_k$.

Semantics of LTL

Assumptions

- Let AP be a set of atomic propositions.
- Let π be a run of Kripke structure $M = (S, T, I, s_0)$.
- Let φ, ψ be syntactically correct LTL formulae.
- Let $p \in AP$ denote atomic proposition.

Semantics

$$
\begin{array}{rcl}
\pi \models p & \text{iff} & p \in I(\pi(0)) \\
\pi \models \neg\varphi & \text{iff} & \pi \not\models \varphi \\
\pi \models \varphi \vee \psi & \text{iff} & \pi \models \varphi \text{ or } \pi \models \psi \\
\pi \models X\,\varphi & \text{iff} & \pi^1 \models \varphi \\
\pi \models \varphi\,U\,\psi & \text{iff} & \exists k.0 \leq k, \pi^k \models \psi \text{ and} \\
 & & \forall i.0 \leq i < k, \pi^i \models \varphi
\end{array}
$$

Semantics of Other Temporal Operators

$$\pi \models F\,\varphi \quad \text{iff} \quad \exists k. k \geq 0, \pi^k \models \varphi$$

$$\pi \models G\,\varphi \quad \text{iff} \quad \forall k. k \geq 0, \pi^k \models \varphi$$

$$\begin{aligned}\pi \models \varphi\,R\,\psi \quad \text{iff} \quad & (\exists k. 0 \leq k, \pi^k \models \varphi \wedge \psi \text{ and} \\ & \forall i. 0 \leq i < k, \pi^i \models \psi) \\ & \text{or } (\forall k. k \geq 0, \pi^k \models \psi)\end{aligned}$$

$$\begin{aligned}\pi \models \varphi\,W\,\psi \quad \text{iff} \quad & (\exists k. 0 \leq k, \pi^k \models \psi \text{ and} \\ & \forall i. 0 \leq i < k, \pi^i \models \varphi) \\ & \text{or } (\forall k. k \geq 0, \pi^k \models \varphi)\end{aligned}$$

LTL Model Checking

Verification Employing LTL

- System is viewed as a set of runs.
- System is satisfies LTL formula if and only if all system runs satisfy the formula.
- In other words, any run violating the formula is a witness that the system does not satisfy the formula.

Lemma

- If a finite state system does not satisfy an LTL formula then this may be witnessed with a **lasso-shaped** run.
- Run π is lasso-shaped if $\pi = \pi_1 \cdot (\pi_2)^\omega$, where

$$\pi_1 = s_0, s_1, \ldots, s_k$$
$$\pi_2 = s_{k+1}, s_{k+2}, \ldots, s_{k+n}, \text{ where } s_k \equiv s_{k+n}.$$

- Note that π^ω denotes infinite repetition of π.

Automata-Based Approach to LTL Model Checking

Languages of infinite words

Observation One

- System is a set of (infinite) runs.
- Also referred to as formal language of infinite words.

Observation Two

- Two different runs are equal with respect to an LTL formula if they agree in the interpretation of atomic propositions (need not agree in the states).
- Let $\pi = s_0, s_1, \ldots$, then $I(\pi) \stackrel{def}{\Longleftrightarrow} I(s_0), I(s_1), I(s_2), \ldots$.

Observation Three

- Every run either satisfies an LTL formula, or not.
- Every LTL formula defines a set of satisfying runs.

Reduction to Language Inclusion

Problem Formulation

- Let the system under verification be given as Kripke structure $M = (S, T, I, s_0)$ and system specification as LTL formula φ.
- Does system M satisfies specification φ? ($M \overset{?}{\models} \varphi$)

Reformulation as Language Problem

- Let $\Sigma = 2^{AP}$ be an alphabet.
- Language L_{sys} of all runs of system M is defined as follows.

$$L_{sys} = \{I(\pi) \mid \pi \text{ is a run in } M\}.$$

- Language L_φ of runs satisfying φ is defined as follows.

$$L_\varphi = \{I(\pi) \mid \pi \models \varphi\}.$$

Observation

- System M satisfies specification φ if and only if $L_{sys} \subseteq L_\varphi$.

L_{sys} and L_φ expressed by Büchi automaton

Theorem

- For every LTL formula φ there exists (and can be efficiently constructed) Büchi automaton A_φ such that $L_\varphi = L(A_\varphi)$.
- Vardi and Wolper, 1986

Theorem

- For every Kripke structure $M = (S, T, I, s_0)$ we can construct Büchi automaton A_{sys} such that $L_{sys} = L(A_{sys})$.
- Construction of A_{sys}
  - Let AP be a set of atomic propositions.
  - Then $A_{sys} = (S, 2^{AP}, s_0, \delta, S)$, where $q \in \delta(p, a)$ if and only if $(p, q) \in T \wedge I(p) = a$.

Synchronous Product of Büchi Automata

Theorem

- Let $A = (S_A, \Sigma, s_A, \delta_A, F_A)$ and $B = (S_B, \Sigma, s_B, \delta_B, F_B)$ be Büchi automata over the same alphabet Σ. Then we can construct Büchi automaton $A \times B$ such that $L(A \times B) = L(A) \cap L(B)$.

Construction of $A \times B$

- $A \times B = (S_A \times S_B \times \{0,1\}, \Sigma, (s_A, s_B, 0), \delta_{A\times B}, F_A \times S_B \times \{0\})$
- $(p', q', j) \in \delta_{A\times B}((p, q, i), a)$ for all

$$
\begin{array}{ll}
p' \in \delta_A(p, a) & \\
q' \in \delta_B(q, a) & \\
j = (i+1) \mathit{mod}\, 2 & \text{if } (i = 0 \wedge p \in F_A) \vee (i = 1 \wedge q \in F_B) \\
j = i & \text{otherwise}
\end{array}
$$

Synchronous Product of Büchi Automata – Task

Let

- $L1 = \{w \in \{a, b, c\}^\omega \mid a \in inf(w)\}$
- $L2 = \{w \in \{a, b, c\}^\omega \mid inf(w) = \{b\}\}$
- $L3 = L1 \cap L2$

Find Büchi automata for $L1$, $L2$ and $L3$.

Synchronous Product of Büchi Automata – Simplification

Observation

- For the purpose of LTL model checking, we do not need general synchronous product of Büchi automata, since Büchi automaton A_{sys} is constructed in such a way that $F_A = S_A$, i.e. it has all states accepting.
- For such a special case the construction of product automata can be significantly simplified.

Construction of $A \times B$ when $F_A = S_A$

- $A \times B = (S_A \times S_B, \Sigma, (s_A, s_B), \delta_{A \times B}, S_A \times F_B)$
- $(p', q') \in \delta_{A \times B}((p, q), a)$ for all

$$p' \in \delta_A(p, a)$$
$$q' \in \delta_B(q, a)$$

Reduction to Büchi Emptiness Problem

Theorem

- For every LTL formula φ it holds that $co{-}L(A_\varphi) = L(A_{\neg\varphi})$.
- By $co{-}M$ we denote complement to the set of all words over the alphabet of M.

Reduction of $M \models \varphi$ to the emptiness of $L(A_{sys} \times A_{\neg\varphi})$

- $M \models \varphi \iff L_{sys} \subseteq L_\varphi$
- $M \models \varphi \iff L(A_{sys}) \subseteq L(A_\varphi)$
- $M \models \varphi \iff L(A_{sys}) \cap co{-}L(A_\varphi) = \emptyset$
- $M \models \varphi \iff L(A_{sys}) \cap L(A_{\neg\varphi}) = \emptyset$
- $M \models \varphi \iff L(A_{sys} \times A_{\neg\varphi}) = \emptyset$

Reduction to Accepting Cycle Detection

Theorem

- Büchi automaton $A = (S, \Sigma, s_0, \delta, F)$ accepts a non-empty language if and only if there is a state $s \in F$ and words $w_1, w_2 \in \Sigma^*$ such that $s \in \hat{\delta}(s_0, w_1)$ a $s \in \hat{\delta}(s, w_2)$.
- That is, the graph of Büchi automaton contains a reachable accepting cycle (cycle through an accepting state).

Decision Procedure for $M \models \varphi$?

- Build a product automaton $(A_{sys} \times A_{\neg\varphi})$.
- Check the automaton for presence of an accepting cycle.
- If there is a reachable accepting cycle then $M \not\models \varphi$.
- Otherwise $M \models \varphi$.

Practicals and Homework – 05

Practicals

- Specifying properties with Büchi Automata.
- Specify properties using LTL.
- Model-based verification using DIVINE model checker.

Homework

- Model Peterson's mutual exclusion protocol in ProMeLa.
- State expected LTL properties of Peterson's protocol.
- Verify them using SPIN model checker.