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 after deployment

- Interaction with environment (unexpected input values).
- Interaction with other system components.
- Parallelism (difficult to test).

Model Checking

- Automated verification process for ...
- ... reactive systems.
- ... parallel/distributed systems.

Verification of Reactive and 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 $\lambda x\text{->}x+1$.
- The composition is: $(\lambda x\text{->}x+1)\cdot(\lambda x\text{->}x+1)$.
- $(\lambda x\text{->}x+1)\cdot(\lambda x\text{->}x+1)\ 0 = 2$

Two Different System Runs

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

Consequence of Incomposability

Consequence 1

- In general, we cannot deduce correctness of the system from the correctness of its components.

Consequence 2

- In general, we cannot decompose the problem of checking correctness of the system into subproblems, i.e. to reduce it to the verification of correctness of all the participating system components.

Specification for Reactive/Parallel Programs

Problem of Specification

- The number of inputs and the input values are unknown before a particular execution of a reactive system.
- **How to specify the system properties?**

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.

Deductive Verification for Modal and Temporal Logic

Model checking
- Approach to formal verification of systems.
- Based on the system's state-space exploration.

Assumption
- Requires specification to be described with a formal precision.

Turning into Formal Language
- Use of Modal and Temporal Logics
- Amir Pnueli, 1977

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
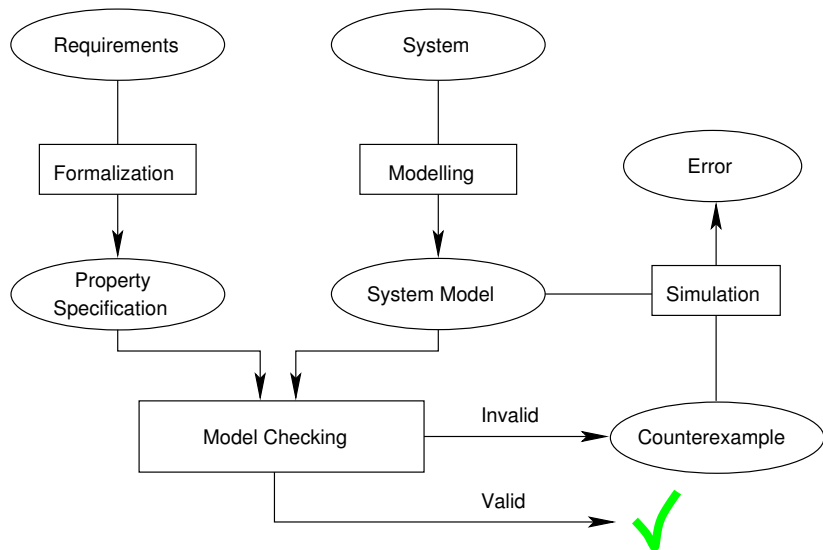
Model Checkers

- 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.

Modelling and Formalization of Verified Systems

Atomic Proposition

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.
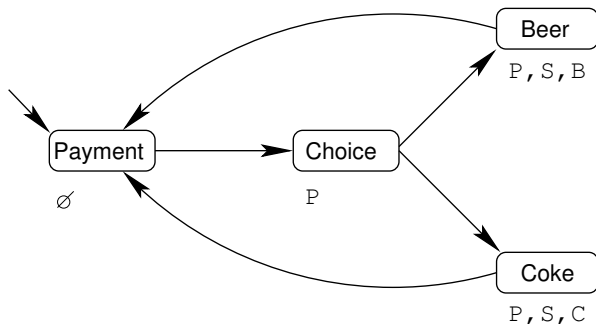
Kripke Structure

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,
    - $I : S \to 2^{AP}$ 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,
    - $\mathcal{L} : T \to Act$ is labelling function.
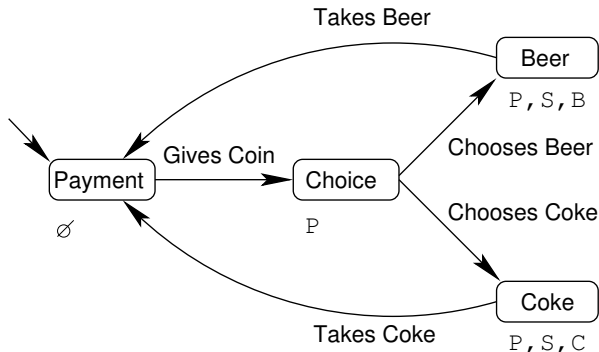
Kripke Structure – Example

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.

An 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
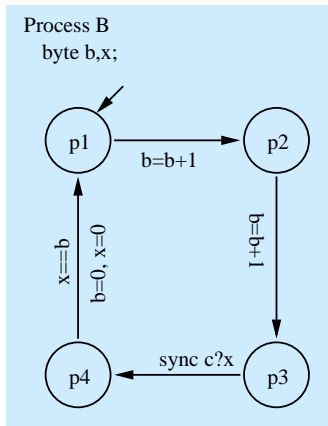
Process B
  byte b,x;

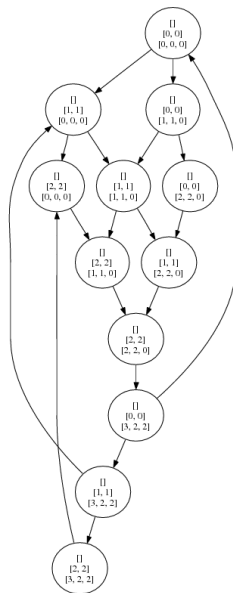# Example of System Described in DVE Language

```
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]
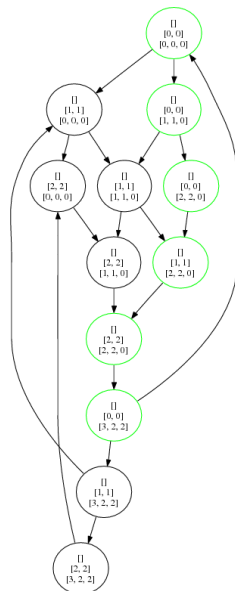
Formalizing System Properties

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.

Automata over infinite words

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,
    - $\delta : S \times \Sigma \to 2^S$ 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 $true \equiv X \vee \neg X$.

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

$$X \varphi : \quad \bullet \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \cdots$$

$$\varphi \, U \, \psi : \quad \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \bullet \cdots$$

$$F \varphi : \quad \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \bullet \cdots$$

$$G \varphi : \quad \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \cdots$$

$$\varphi \, R \, \psi : \quad \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\varphi \wedge \psi}{\bullet} \longrightarrow \bullet \cdots \quad or$$

$$\qquad\qquad \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \cdots$$

$$\varphi \, W \, \psi : \quad \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\psi}{\bullet} \longrightarrow \bullet \cdots \quad or$$

$$\qquad\qquad \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \longrightarrow \overset{\varphi}{\bullet} \cdots$$

Syntax of LTL
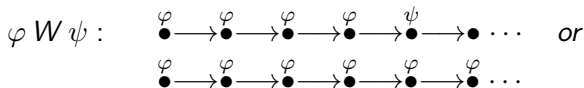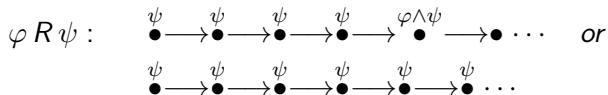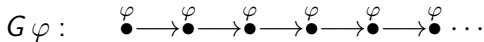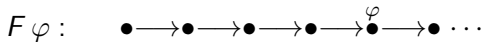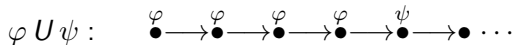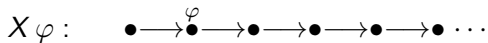
Let AP be a set of atomic propositions.

- 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

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

Alternative syntax

- $F\varphi \equiv \diamond\varphi$
- $G\varphi \equiv \square\varphi$
- $X\varphi \equiv \circ\varphi$

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

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

Semantics of Other Temporal Operators

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

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

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

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

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

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

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

Homework

Homework

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