Software Systems Verification and Validation

Vescan Andreea, PHD, Assoc. Prof.

Faculty of Mathematics and Computer Science
Babeș-Bolyai University

2023-2024

Software Systems Verification and Validation

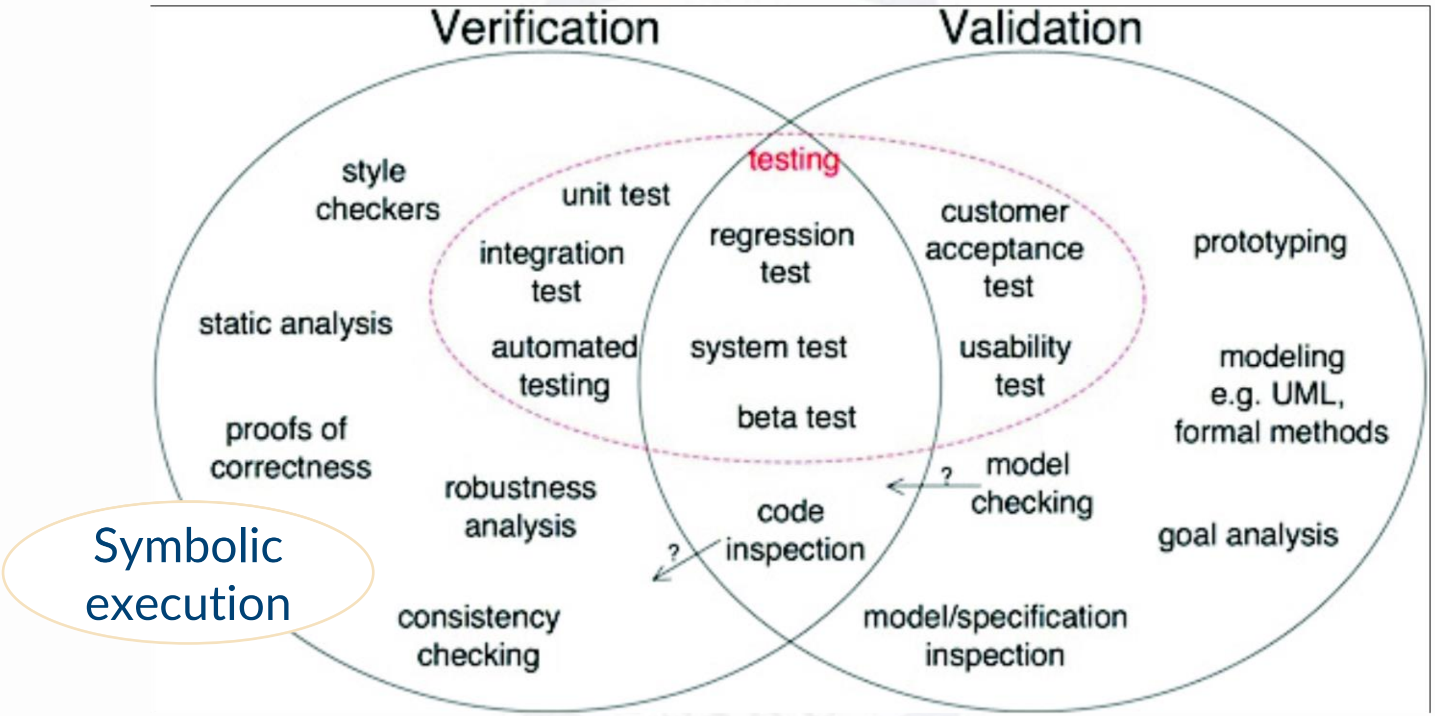
"Tell me and I forget, teach me and I may remember, involve me and I learn."

(Benjamin Franklin)

(Next)/Today Lecture

- Symbolic execution
- Model checking

What we will learn!



- http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/

Outline

Model checking

- System verification
- Model checking
- Transition system
- Linear-Time Properties
- Linear-Time Logic
- Computation Tree Logic

Spin Model Checker

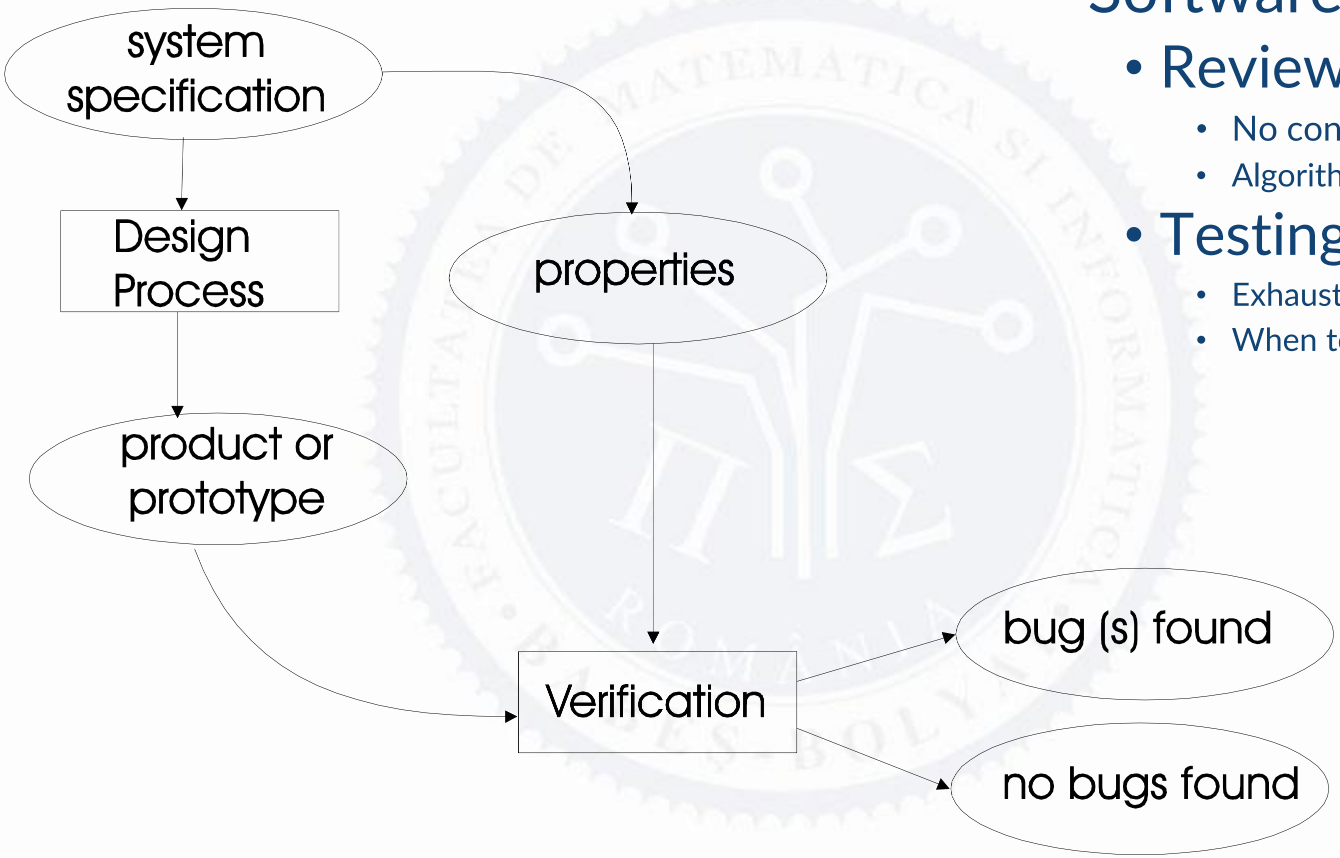
- Spin
- Promela Model
  - Statements
  - Examples
- Concurrency and Interleaving Semantics
  - Examples
- Linear Temporal Logic
  - Examples
- JSpin

- Questions

System verification (1)

- Information and Communication Technology (ICT)
- Correct ICT systems
  - It is all about money.
  - It is all about safety.
- Reliability of the ICT systems
  - Interactive systems - concurrency & nondeterminism
  - Pressure - to reduce system development time
- System verification techniques

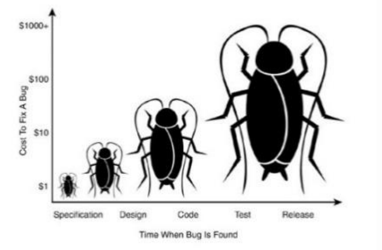
System verification (2)

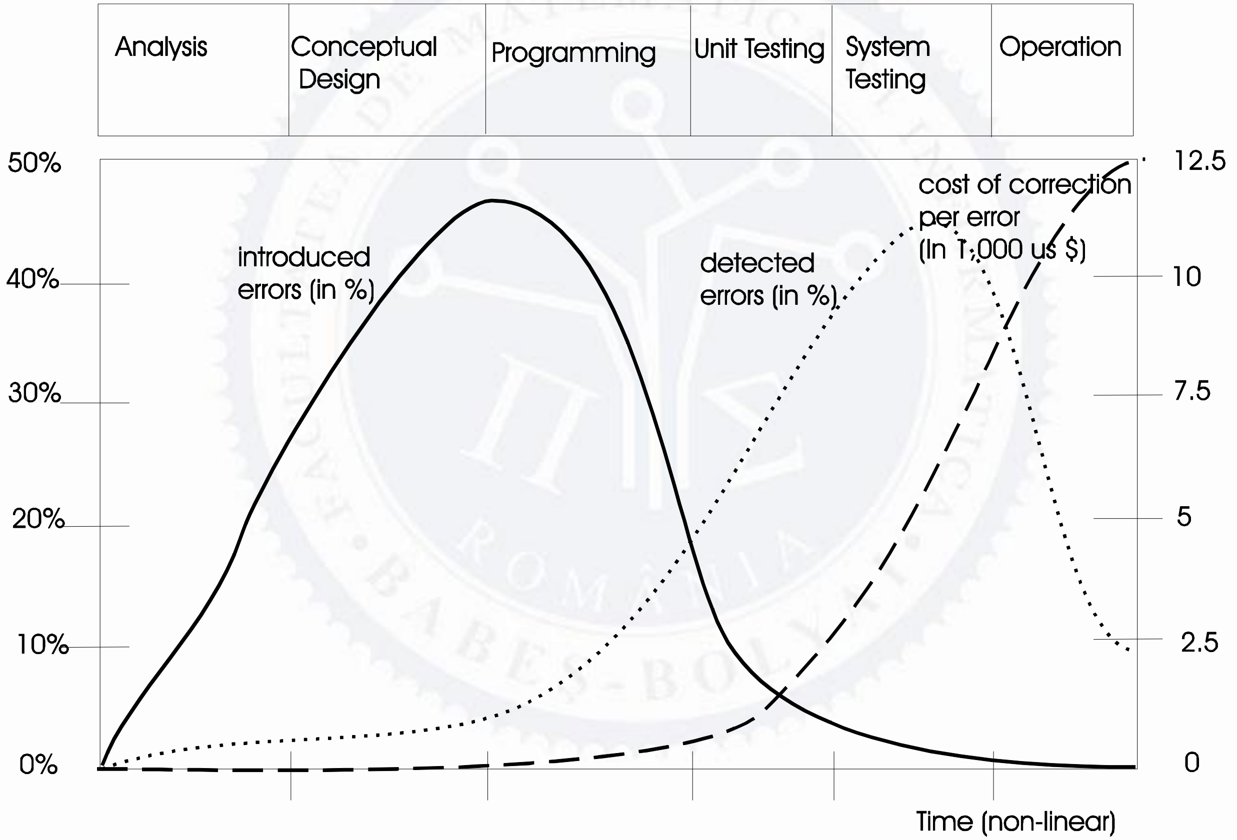


- Software verification
  - Review
    - No concurrency defects
    - Algorithm defects
  - Testing
    - Exhaustive testing?
    - When to stop?

System verification (2)

- Catching software errors: the sooner the better





Model checking (1)

Formal methods

- More time and effort spend on verification than on construction
  - in software/hardware design of complex systems.
- The role of formal methods:
  - To establish system correctness with mathematical rigor.
  - To facilitate the early detection of defects.
- Verification techniques
  - Testing – small subset of paths is treated
  - Simulation - restrictive set of scenarios in the model
  - Model checking - exhaustive exploration
- **Remark.** Any verification using **model-based techniques** is only as good as the model of the system.

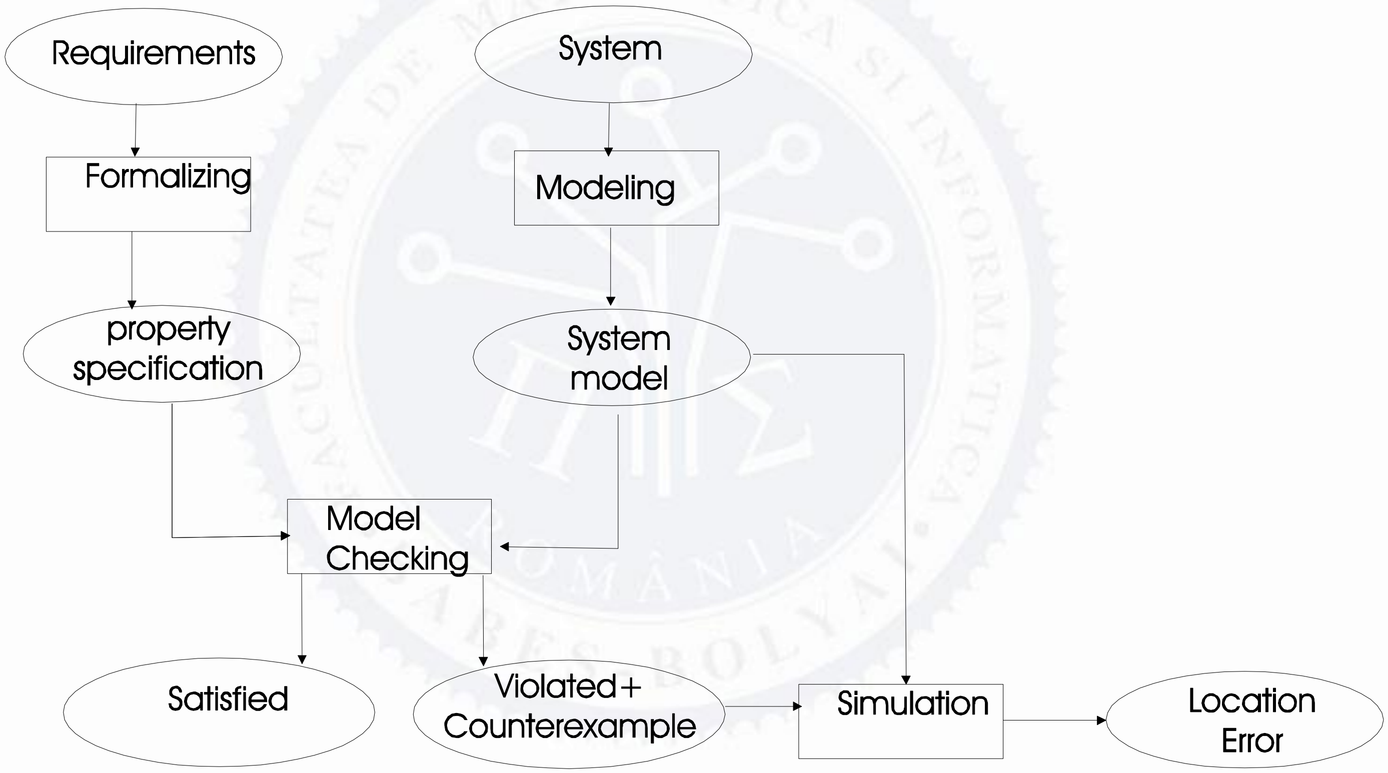
Model checking (1)

Formal methods

- Mechanical Engineering is like looking for a black cat in a lighted room.
- Chemical Engineering is like looking for a black cat in a dark room.
- Software Engineering is like looking for a black cat in a dark room in which there is no cat.
- Systems Engineering is like looking for a black cat in a dark room in which there is no cat and some-one yells, “I got it!”

Model checking (2)

Approach



Model checking (3)

Characteristics

- Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.
- The model checking process
  - Modeling phase
    - model the system under consideration
    - formalize the property to be checked.
  - Running phase
  - Analysis phase
    - property satisfied?
    - property violated?

Model checking (4)

Strengths	Weaknesses
• General verification approach • Supports partial verification • Provides diagnostic information • Potential “push-button” technology • Increasing interest by industry • Easily integrated in existing development cycles	• Appropriate to control-intensive applications • Its applicability is subject to decidability issues • It verifies a system model • Checks only stated requirements • Suffers from the state-space explosion problem • Requires some expertise

Transition system (1)

Definition

- Transition systems - used in computer science as models to describe the behavior of the systems.
- Transition systems - directed graphs:
  - Nodes - represent states;
  - Edges - model transitions, i. e. state changes.
- A Transition System (TS) is tuple (S, Act, $\rightarrow$, I, Ap, L), where
  - S is a set of states,
  - Act is a set of actions,
  - $\rightarrow \subseteq S \times Act \times S$ is a transition relation,
  - $I \subseteq S$ is a set of initial states,
  - AP is a set of atomic propositions, and
  - $L : S \rightarrow 2^{AP}$ is a labeling function.
- TS is called finite if S, Act and AP are finite.

Transition system (2)

Remark

- Intuitive behavior of a transition system
  - Initial state $s_0 \in I$
  - Using the transition relation $\rightarrow$ the system evolves
  - Current state s, a transition $s \xrightarrow{\alpha} s'$ is selected *nondeterministically*
  - The selection procedure is repeated and finishes once a state is encountered that has no outgoing transitions.
- The labeling function L relates a set $L(s) \in 2^{AP}$ at atomic propositions to any state s. $L(s)$ intuitively stands for exactly those atomic propositions $a \in AP$ which are satisfied by state s.
- Given that ϕ is a propositional logic formula, then s satisfies the formula ϕ if the evaluation induced by $L(s)$ makes the formula ϕ true.

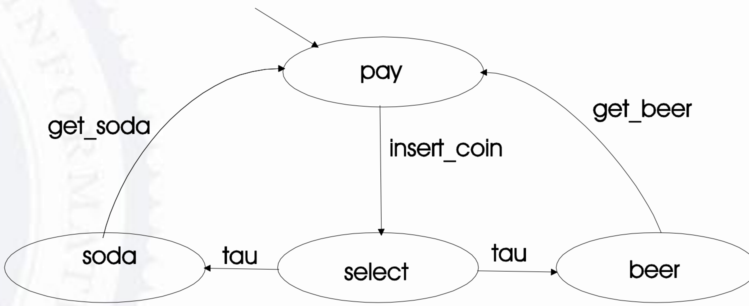
  $s \models \phi$ iff $L(s) \models \phi$.

Transition system (3)

Example

Beverage Vending Machine

- $S = \{pay, select, soda, beer\}$, $I = \{pay\}$
- $Act = \{insert\_coin, get\_soda, get\_bear, \tau\}$
- Example transitions: $pay \overset{insert\_coin}{\rightarrow} select$, $beer \overset{get\_beer}{\rightarrow} pay$
- Atomic propositions depends on the properties under consideration.
  A simple choice - to let the state names act as atomic propositions, i. e. $L(s) = \{s\}$.
  "The vending machine only delivers a drink after providing a coin."
  $AP = \{paid, drink\}$, $L(pay) = \emptyset$, $L(soda) = L(beer) = \{paid, drink\}$, $L(select) = \{paid\}$.



Linear-Time Properties

- **Deadlock** – if the complete system is in a terminal state, although at least one component is in a (local) nonterminal state.
  - A typical deadlock scenarios occurs when components mutually wait for each other to progress.

- **Safety properties** = “nothing bad should happen”.
  - The number of inserted coins is always at least the number of dispensed drinks.
  - A typical safety property is deadlock freedom
  - Mutual exclusion problem – “bad” = more than one process is in the critical section

- **Liveness properties** = “something good will happen in the future”.
  - Mutual exclusion problem – typical liveness properties assert that:
    - (eventually) – each process will eventually enter its critical section
    - (repeated eventually_ = each process will enter its critical section infinitely often
    - (starvation freedom) – each waiting process will eventually enter its critical section

- **Remark**
  - **Safety properties** - are violated in finite time (a finite system run)
  - **Liveness properties** – are violated in infinite time (by infinite system runs)

Temporal Logic

- **Propositional temporal logics -** extensions of propositional logic by temporal modalities.

- The elementary temporal modalities that are present in most temporal logics include the operators
  - “**eventually**” (eventually in the future) -
  - “**always**” (now and forever in the future –

- The nature of time in temporal logics can be either **linear** or **branching.**

- The adjective “temporal”
  - specification of the relative order of events
  - does not support any means to refer to the precise timing of events

Linear-Time Logic (1)

Syntax of LTL

- Construction of LTL formulae in LTL - ingredients:
  - atomic propositions $a \in AP$, (stands for the state label a in a transition system)
  - boolean connectors like conjunction $\wedge$ and negation $\neg$,
  - basic temporal modalities "next" $\bigcirc$ and "until" $\bigcup$.
- LTL formulae over the set AP of atomic proposition are formed according to the following grammar:
  $\varphi ::= true | a | \varphi_1 \wedge \varphi_2 | \neg\varphi | \bigcirc \varphi | \varphi_1 \bigcup \varphi_2$, where $a \in AP$.

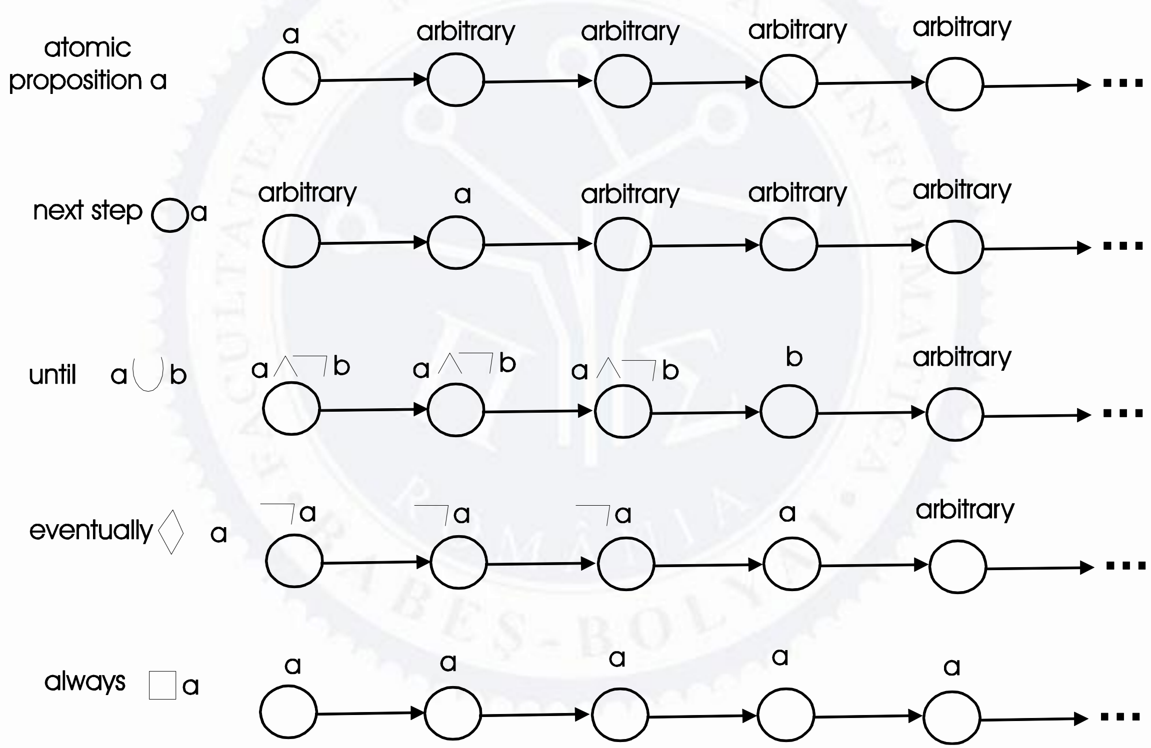
Linear-Time Logic (2)

LTL temporal modalities

- The until operator allows to derive the temporal modalities $\Diamond$ ("eventually", sometimes in the future) and $\Box$ ("always", from now on forever) as follows:
  - $\Diamond\varphi = \text{true} \bigcup \varphi$.
  - $\Box\varphi = \neg\Diamond\neg\varphi$.
- By combining the temporal modalities $\Diamond$ and $\Box$, new temporal modalities are obtained:
  - $\Box\Diamond\varphi$ - "infinitely often φ."
    at any moment j there is a moment i $i \geq j$ at which an a state is visited
  - $\Diamond\Box\varphi$ - "eventually forever φ."
    from some moment j on, only a-states are visited.

Linear-Time Logic (3)

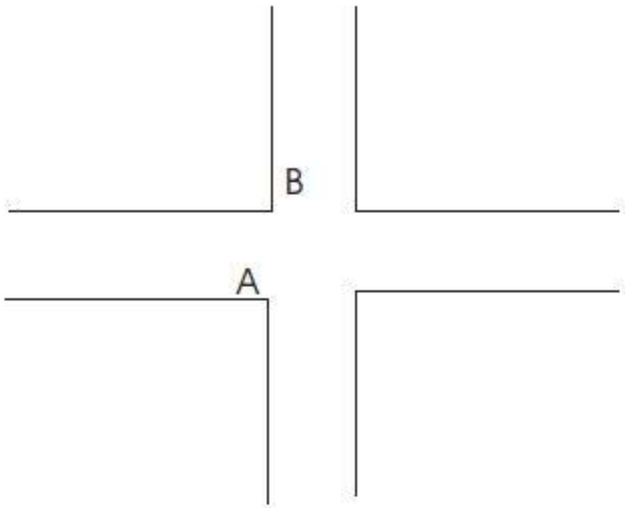
Intuitive meaning of temporal modalities



Linear-Time Logic (4)

LTL semaphore example

- $\Box(\neg(A = green \wedge B = green))$
  - A and B can not be simultaneously green.
- $\Box(A = yellow \rightarrow A = red)$
  - If A is yellow eventually will become red.
- $\Box(A = yellow \rightarrow \bigcirc(A = red))$
  - If A is yellow then it will be red into the next state.
- $\Box(\neg(B = green) \bigcup (A = red))$
  - B will not be green until A changes in red.



Computation Tree Logic (1)

Syntax of CTL

- Construction of CTL formulae:
  - as in LTL by the next-step and until operators,
  - must be not combined with boolean connectives
  - no nesting of temporal modalities is allowed.
- CTL formulae over the set AP of atomic proposition are formed according to the following grammar:
  $\phi ::= \text{true } |a|\phi_1 \wedge \phi_2|\neg\phi|\exists\phi|\forall\phi$, where $a \in \; AP \; and \; \varphi$ is a path formula.
- CTL path formulae are formed according to the following grammar:
  $\varphi ::= \bigcirc\phi|\phi_1 \bigcup \phi_2$, where $\phi, \phi_1 and \phi_2$ are state fromulae.

Computation Tree Logic (2)

CTL - state and path formulae

- CTL distinguishes between state formulae and path formulae:
  - State formulae express a property of a state.
  - Path formulae express a property of a path, i.e. an infinite sequence of states.
- Temporal PATH operators $\bigcirc$ and $\bigcup$
  - $\bigcirc\phi$ holds for a path if ϕ holds in the next state of the path;
  - $\phi \bigcup \psi$ holds for a path if there is some state along the path for which ψ holds, and ϕ holds in all

    states prior to that state.
- Path formulae $\Rightarrow$ state formulae by prefixing them with
  - path quantifier $\exists$ (pronounced "for some path");
    $\exists\phi$ - holds in a state if there exists some path satisfying ϕ that starts in that state.
  - path quantifier $\forall$ (pronounced "for all paths".)

    $\forall\phi$-holds in a state if all paths that start in that state satisfy ϕ.

Outline

Model checking

- System verification
- Model checking
- Transition system
- Linear-Time Properties
- Linear-Time Logic
- Computation Tree Logic

Spin Model Checker

- Spin
- Promela Model
  - Statements
  - Examples
- Concurrency and Interleaving Semantics
  - Examples
- Linear Temporal Logic
  - Examples
- JSpin

- Questions

Model checking

Spin

- Developed at Bell Labs.
- In 2002, recognized by the ACM with Software System Award.
- SPIN (= Simple Promela Interpreter)
- is a tool for analyzing the logical consistency of concurrent systems
- Concurrent systems are described in the modelling language called Promela (= Protocol/Process Meta Language)

Promela

- Promela (= Protocol/Process Meta Language)
- allows for the dynamic creation of concurrent processes.
- communication via message channels can be defined to be
  - synchronous (i.e. rendezvous),
  - asynchronous (i.e. buffered).

Promela Model

- Promela model consist of:
  - type declarations
  - channel declarations
  - variable declarations
  - process declarations
  - **[init process]**
- A process type (**proctype) consist of**
  - a name
  - a list of formal parameters
  - local variable declarations
  - Body

- A process
  - is defined by a **proctype definition**
  - executes concurrently with all other processes, independent of speed of behaviour
  - communicate with other processes
    - using global (shared) variables
    - using channels
- There may be several processes of the same type.
- Each process has its own local state:
  - process counter (location within the **proctype)**
  - contents of the local variables

Promela Model - Statements

- The body of a process consists of a sequence of statements.
- A statement is either
  - executable: the statement can be executed immediately.
  - blocked: the statement cannot be executed.
- An assignment is always executable.
- An expression is also a statement; it is executable if it evaluates to non-zero
- The **skip statement is always executable.**
  - "does nothing", only changes process' process counter
- A **printf statement is always executable (but is not** evaluated during verification, of course).
- **assert(<expr>);**
  - The **assert-statement is always executable.**
  - If **<expr> evaluates to zero, SPIN will exit with an error, as**
- the **<expr> "has been violated".**
  - The **assert-statement is often used within Promela models,**
- to check whether certain properties are valid in a state.

Examples (01 Simple Examples)

- ReversingDigits.pml
  - Check
  - Random
- DiscriminantOfQuadraticEquation.pml
  - Check
  - Random
- NumberDaysInMonth.pml
  - Check
  - Random
- MaximumNondeterminism.pml
  - Check
  - Random
  - “Branch 1” and “Branch 2”
- Maximum –second example-MaximumIfElse.pml
  - Check
  - Random
- GCD.pml
  - Check
  - Random
- IntegerDivison01.pml
  - Check
  - Random

Concurrency and Interleaving Semantics

02 Concurrency and interleaving semantics

- Promela processes execute concurrently.
  - Non-deterministic scheduling of the processes.
  - Processes are interleaved (statements of different processes do not occur at the same time).
  - exception: rendez-vous communication.
- All statements are atomic; each statement is executed without interleaving with other processes.
- Each process may have several different possible actions enabled at each point of execution - only one choice is made, non-deterministically.
- InterleavingStatements.pml
  - Check
  - Random
  - 6 possibilities of the execution
    - n1,p,n2,q;
    - n1,n2,p,q;
    - n1,n2,q,p;
    - n2,q,n1,p;
    - n2,n1,q,p;
    - n2,n1,p,q.
  - Interactive simulation – Interactive button
- InterferenceBetweenProcesses.pml
- InterferenceBetweenProcessesDeterministic.pml

Examples (03 Critical section)

- CriticalSection_Incorrect.pml
  - both processes – in the critical section
- CriticalSection_MutualExclusion.pml – not satisfied
  - Mutual exclusion – at most one process is executing its critical section at any time.
- CriticalSection_With_Deadlock.pml
  - Blocking on an expression – user Interactive simulation
  - Absence of deadlock – it is impossible to reach a state in which come processes are trying to enter their critical sections, but no process is successful.
- CriticalSection_SolutionAtomic.pml
  - The atomic sequence may be blocked from executing, but once it starts executing, both statements are executed without interference from the other process.

Linear Temporal Logic

- Temporal logic formulae can specify both safety and liveness properties.

- LTL ≡ propositional logic + temporal operators

  `[]P` always P

  `<>P` eventually P

  `P U Q` P is true until Q becomes true

Examples (04 LTL examples)

- CriticalSection_MutualExclusionLTL.pml
  - LTL formula:
    - []mutex
  - Translate
  - Verify
- CriticalSection_MutualExclusionLTL02.pml
  - LTL formula:
    - []mutex
  - Translate
  - Verify
- CriticalSection_With_Starvation.pml
  - LTL formula:
    - <>csp
  - Translate
  - Acceptance
  - Verify

Channels in Promela 05

Channels

- A channel in Promela = a data type with two operations:
  - send
    - The send statement consists of a channel variable followed by **an exclamation point** and then a sequence of expressions whose number and types match the message type of the channel.
  - receive
    - The receive statement consists of a channel variable followed by **question mark** and a sequence of variables.
- Every channel has associated with it a message type.
- The message type that specifies the structure of each message that can be send on the channel as a sequence of fields).

  Chan ch = [capacity] of {typname, …, typename}
- There are two types of channels with different semantics:
  - Rendezvous channels of capacity 0
  - Buffered channels of capacity greater than 0
- Examples
- **Client-server-channels.plm**

Rendezvous Channels in Promela
05 Channels (Book [2]: pages 107-109)

- Rendezvous channel – with capacity 0.
  - The transfer of the message from the sender (a process with a send statement) to the receiver (a process with the receive statement) is synchronous and is executed as a single atomic operation.
- Examples
- **Simple-Rendezvous.pml**
  - The rendezvous is one atomic operation; even if there were other processes, no interleaving could take place between the execution of the send statement and the receive statement.

Traffic-Pedestrian 06 Channels

- Examples

- **PromelaMarryMe_Simple.pml**
- **PromelaMarryMe.pml**
- **traffic_pedestian.pml**

JSpin

- http://spinroot.com/
- Installation JSpin

  http://jspin.software.informer.com/5.0/

Outline

Mentimeter

- Static analysis, Testing, Symbolic execution
- Conventional vs Symbolic execution
- Symbolic execution for sequential, alternative, repetitive structures
  - Sequential structure execution
  - Alternative structure execution
  - Repetitive structure execution
- Symbolic Execution Tree
  - Symbolic Execution Tree
  - Properties

- System verification
- Model checking
- Transition system
- Linear-Time Properties
- Linear-Time Logic
- Computation Tree Logic

- Spin
- Promela Model
  - Statements
  - Examples
- Concurrency and Interleaving Semantics
  - Examples
- Linear Temporal Logic
  - Examples
- JSpin

References

- [1] Baier Christel, Katoen Joost-Pieter, Principles of Model Checking , ISBN 9780262026499, The MIT Press, 2008
  - Chapter 1 - System verification, Chapter 2 – Modelling Concurrent systems (pag. 19-20), Chapter 3 (pag. 89, 107, 120-121), Chapter 5 – Linear Temporal Logic (pag. 229-233), Chapter 6 – Computation Tree Logic (pag. 313-323)

- [2] Ben-Ari, Mordechai, Principles of the Spin Model Checker, ISBN 978-1-84628-770-1, Springer-Verlag London, 2008

Next Lecture

- Fundmore Invited Lecture

QA&QC DURING
THE SOFTWARE DEVELOPMENT LIFE CYCLE

Presenters:	**Monday**
Roxana Soporan	**22 April 2024, 8-10 am**
Andra Banto	**Room: 2/I (Main Building)**

Software Systems Verification and Validation

"Tell me and I forget, teach me and I may remember, involve me and I learn."

(Benjamin Franklin)