Systems Security Group

Lancaster University

Introduction to Model Checking

Learning Objectives

- Learn about formal verification and the basics of model checking
- Familiarise with transition systems
- Learn the type of properties that can be verified and how these could be defined

Principal Methods for Verifying Complex Systems (1)

- Testing
  - Performed on the system itself
  - How about testing distributed systems?
  - Proves the existence of bugs, but not their absence
- Simulation
  - A finite number of user-defined system trajectories meet the desired specification
  - How about completeness?

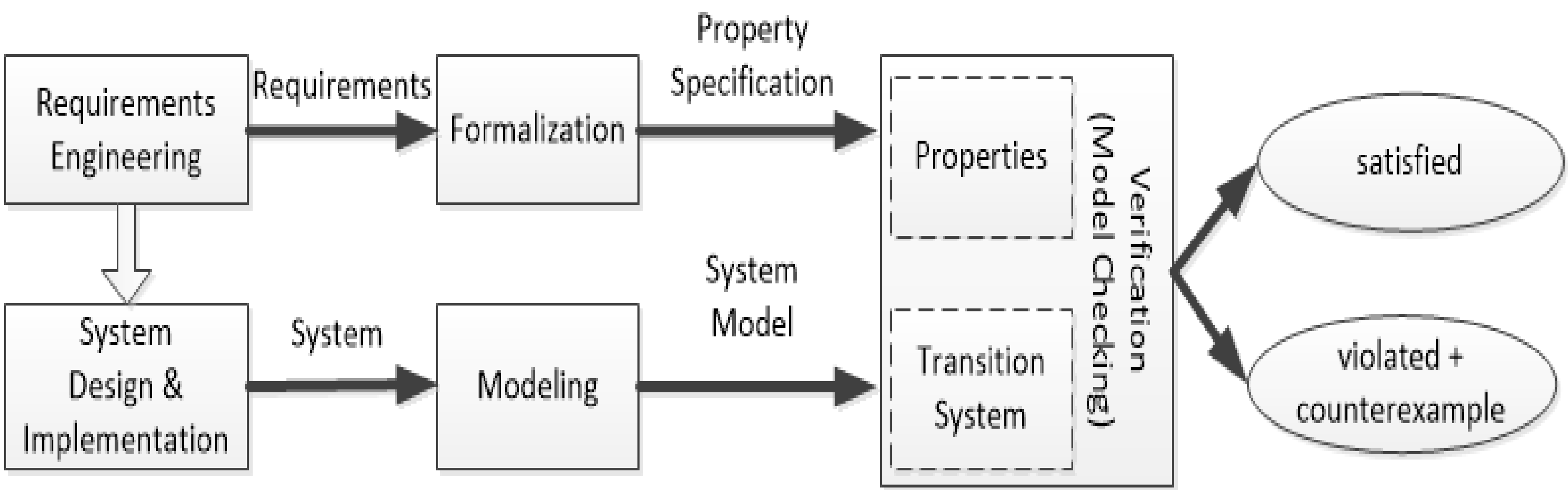
Principal Methods for Verifying Complex Systems (2)

- Deductive verification
  - Manual mathematical proof of correctness of a model of a system
  - Cost?
  - Skills?
- Model checking
  - Automated technique
  - Given a finite-state model of a system and a formal property it checks systematically whether the property holds or not for that model

The Model Checking Process (1)

- Modelling phase
  - Model a system using a language
  - Define the formal properties to be checked
- Running phase
  - The model checker checks the validity of properties in the system model
- Analysis phase
  - Property satisfied … check for next
  - Property violated … look at counterexamples and refine the model

The Model Checking Process (2)



Systems Security Group | Lancaster University

Pros of Model Checking

- It is a general verification approach
- It supports partial verification
- It is not vulnerable to the likelihood that an error is exposed
- It provides diagnostic information
- Highly automated
- Has a sound and mathematical underpinning

Systems Security Group | Lancaster University

Cons of Model Checking

- State-space explosion problem
- Not suited for infinite-state system or reasoning of abstract data types
- Verifies the system model and not the actual system
- Requires some expertise
- Model checkers are still software having defects

A Few Tools…

- nuXmv: a new symbolic model checker
  https://nuxmv.fbk.eu

- TLA+: a high-level language for modelling programs and systems
  https://lamport.azurewebsites.net/tla/tla.html

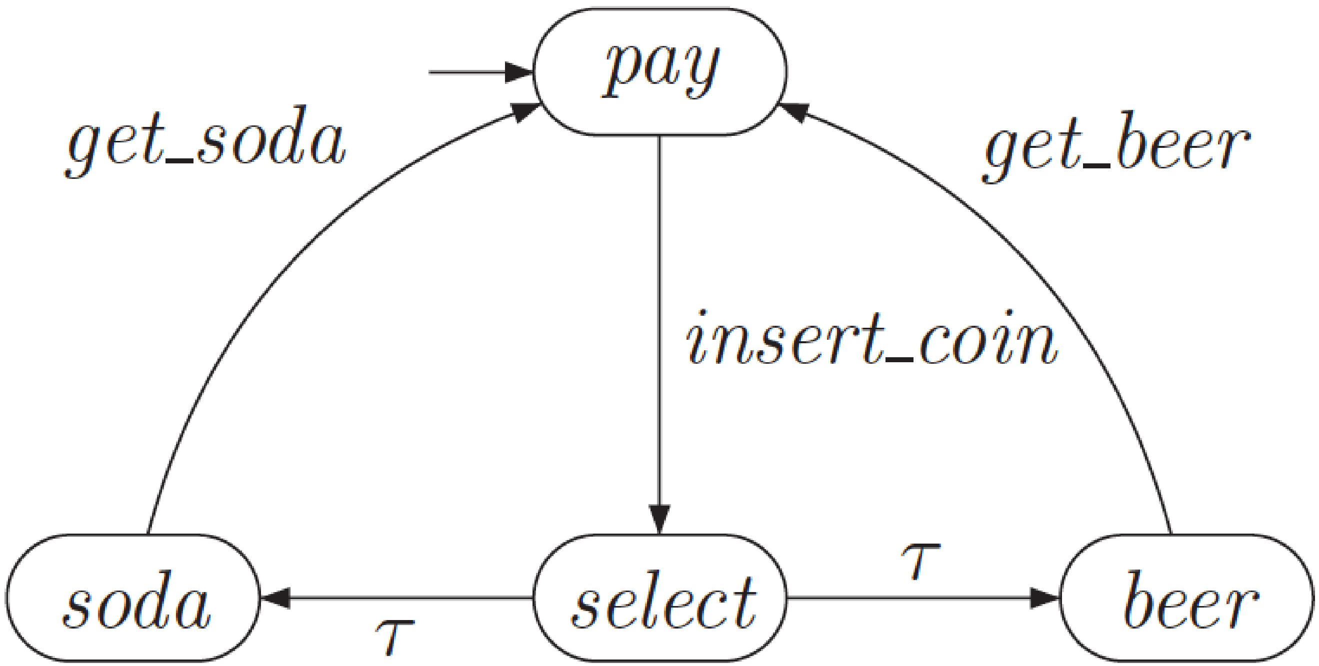
- Alloy: language and analyser for software verification
  https://alloytools.org/

- And many more…

Transition System (TS)

- A TS is a tuple (S, Act, →, I, AP, L) where
  - S is a set of states
  - Act is a set of actions
  - → subset of S x Act x S is a transition relation
  - I subset of S is the set of initial states
  - AP is a set of atomic propositions for the states
    - Temporal characteristics to express simple know facts about the states of the system under consideration
  - L:S → 2^{AP} is a labelling function
- TS is called finite if S, Act and AP are finite.

A TS for a Vending Machine



TS Elements

- The state space $S = \{pay, select, soda, beer\}$
- Initial state $I = \{pay\}$
- Actions $Act = \{insert\_coin, get\_soda, get\_beer, \tau\}$
- $AP = \{paid, drink\}$

  $L(pay) = \{\}$, $L(select) = \{paid\}$

  $L(soda) = L(beer) = \{paid, drink\}$

What type of properties can we verify?

Safety properties

- Safety characterised as:

  'nothing bad should happen'
- Examples of safety properties to verify
  - Mutual exclusion
  - Deadlock freedom
- Safety properties as *'invariants'*
  - Considering a series of states in a system (behaviour) a given condition Φ is required to hold for all reachable states.

Liveness properties

- Liveness characterised as:
  'something good will happen in the future'

- Complementary properties to the safety ones

- Certain events occur infinitely often

- Examples
  - Each process will eventually enter its critical section
  - Each waiting process will eventually enter its critical section

Fairness

- Fairness assumptions rule out infinite behaviour (considered unrealistic)
- Often necessary to establish liveness properties
- Example on process fairness
  - Let's consider *N* processes P_1 ... P_n that request a service
  - What's an unfair and fair strategy?

Fairness constraints

- Unconditional fairness
  ‘Every process gets its turn infinitely often’

- Weak fairness
  ‘Every process that is continuously enabled from a certain time instant on gets its turn infinitely often’

- Strong fairness
  ‘Every process that is enabled infinitely often gets its turn infinitely often’

Other type of properties

- Functional correctness
  - Is the system behaving as it should?
- Reachability
  - Is it possible to reach to a certain state?
- Real-time properties
  - Is the system responding in a timely manner?

How do we specify properties?

- Use of logical formalisms, e.g., temporal logic
- Linear Temporal Logic (LTL)
  - Provides a logical formalism for specifying linear time properties
- Computation Tree Logic (CTL)
  - Branching temporal logic for specifying properties of a system

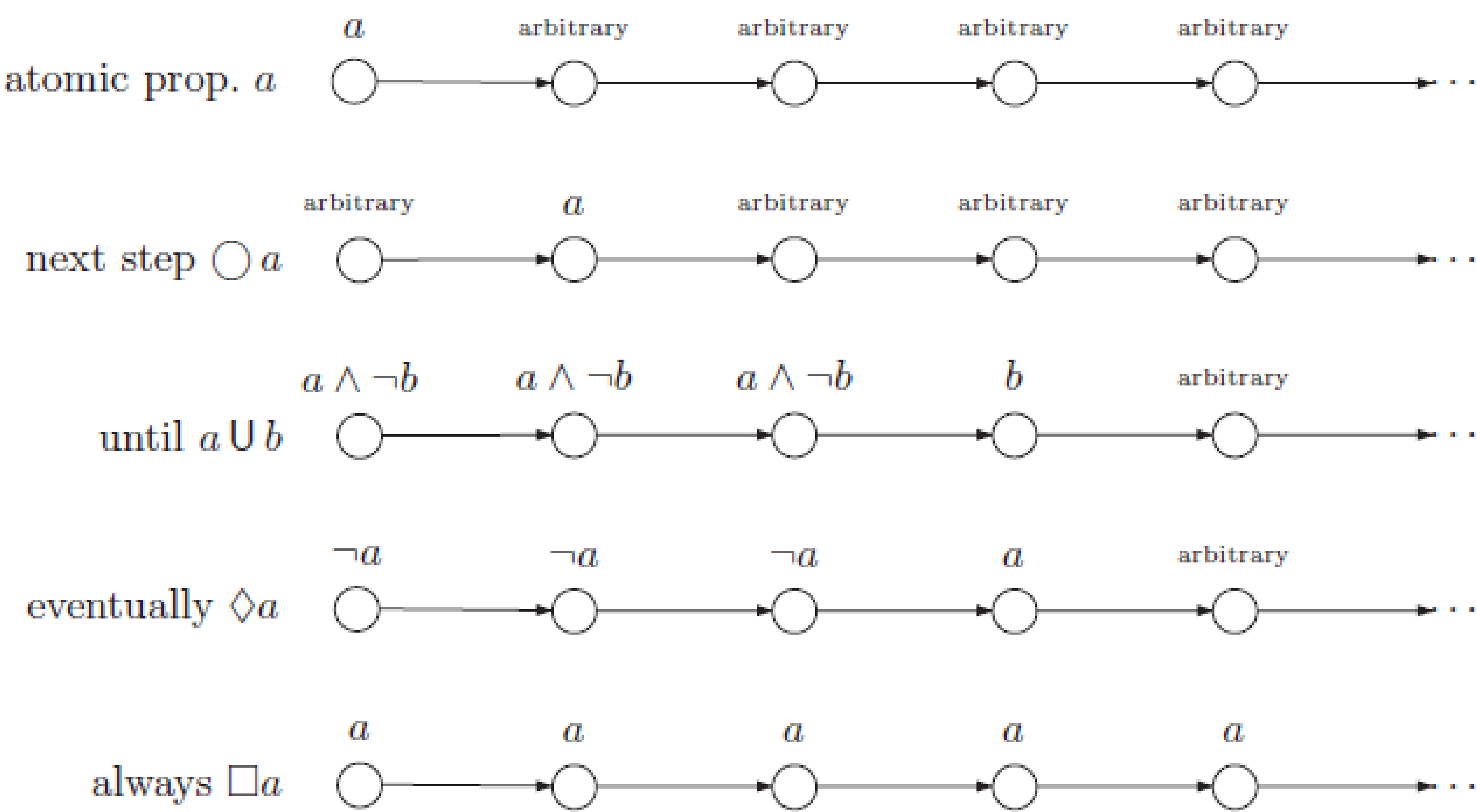
Linear Time Logic

- Make use of atomic propositions: $a \in AP$
- Boolean connectors: $\wedge, \neg$
- Elementary temporal modalities
  - $\diamond$ ‘eventually’ in the future
  - $\square$ ‘always’, now and forever in the future
- Unary prefix operator for ‘next’: O, e.g., $\circ\, \varphi$
- Binary infix operator for ‘until’: U, e.g., $\varphi_1 \cup \varphi_2$
- Syntax of LTL

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \circ\, \varphi \mid \varphi_1 \cup \varphi_2$$

where $\alpha \in \mathrm{AP}$

Semantics of temporal modalities



Hands on…

- How to model the vending machine?
- What type of properties can we verify?

Vending machine in nuXmv #1

-- Define variables

```
VAR
    state: {pay, select, soda, beer};
-- Indicates whether a coin is inserted
    insert_coin: boolean;
-- Indicates whether a soda is dispensed
    get_soda: boolean;
-- Indicates whether a beer is dispensed
    get_beer: boolean;
```

Systems Security Group | Lancaster University

Vending machine in nuXmv #2

-- Initialize variables in the SM

```
ASSIGN
-- Initial state is 'pay'
    init(state) := pay;
-- Initially, no coin is inserted
    init(insert_coin) := FALSE;
-- Initially, no soda is dispensed
    init(get_soda) := FALSE;
-- Initially, no beer is dispensed
    init(get_beer) := FALSE;
```

Vending machine in nuXmv #3

```
-- Define the next state and output values for each
state
next(state) :=
    case
        state = pay & !insert_coin : pay;
        state = pay & insert_coin : select;
        state = select & !insert_coin : select;
        state = select & insert_coin : {soda, beer};
        state = soda : pay;
        state = beer : pay;
        TRUE : state;
    esac;
```

Vending machine in nuXmv #4

```
next(insert_coin) :=
    case
-- A coin is inserted for the first time in 'select' state
        state = select & !insert_coin : TRUE;
-- Coin remains inserted
        insert_coin : insert_coin;
        TRUE : FALSE;
    esac;
```

Vending machine in nuXmv #5

```
next(get_soda) :=
    case
-- Soda is dispensed when in 'select' state with a coin inserted
        state = select & insert_coin : TRUE;
         TRUE : FALSE;
    esac;
next(get_beer) :=
    case
-- Beer is dispensed when in 'select' state with a coin inserted
        state = select & insert_coin : TRUE;
        TRUE : FALSE;
    esac;
```

Vending machine in nuXmv #6

```
-- Atomic propositions
DEFINE
    paid := insert_coin;
    drink := get_soda | get_beer;
-- LTL properties

LTLSPEC G ((state = soda | state = beer) -> F(paid & drink));

LTLSPEC G (!paid -> (F drink));
```

nuXmv demo…

Questions?

References

- Baier, C., & Katoen, J. P. (2008). Principles of model checking. MIT press. Chapters 1, 2, 3, 5, 6.
- https://nuxmv.fbk.eu/downloads/nuxmv-user-manual.pdf