Model Checking
Model Checking Process

For increasing our confidence in the correctness of the model:
- Verification: The model satisfies important system properties
- Debugging: Study counter-examples, pinpoint the source of the error, correct the model, and try again
Digicode

- Consider a program that checks the input given to a bike lock
- Assume we have just three possible entries: A, B, C.
- The bike lock opens only if the combination ABA is digited
- This program can be represented by an automaton with 4 states and 9 transitions
typedef enum State{s1, s2, s3, s4} State;
int main(){
    State s=s1;
    while (true){
        read(x);
        switch (s) {
            case s1:
                if (x==A) then s=s2; break;
            case s2:
                if (x==B) then s=s3;
                else if (x!=A) then s=s1;
                break;
            case s3:
                if (x==A) then {s=s4; return 1;}
                else s=s1;
                break;
            default: break;
        }
    }
}
Digicode
Executions

• An execution is a sequence of states that describes a possible evolution of the system.

• For instance, 1121, 12234, 112312234 are possible executions of the digicode.

• The possible executions of the digicode are:
  1
  11,12
  111,112,121,122,123
  1111,1112,1121,1122,1123,1211,1212,1221,1222,1223,1231,1234
  ...


The execution tree
Properties

• Each state of the automaton is associated with some elementary properties that are true when the system is in that state.

• For instance, the property “the bike lock is open” holds on state 4, but it does not hold in the states 1, 2 and 3.

• We would like to show some properties like
  – If the bike lock opens, then the last three letters that have been digited are ABA, in this order
  – If the input contains a sequence of letters that ends with ABA, the bike lock opens.
Atomic formulas

• In our digicode example, the basic formulas are
  – \( P_A \): the last input digited is A
  – \( P_B \): the last input digited is B
  – \( P_C \): the last input digited is C
  – \( \text{pred}_1 \): the previous state is state 1
  – \( \text{pred}_2 \): the previous state is state 2
  – \( \text{pred}_3 \): the previous state is state 3
Adding atomic formulas to the automaton
• Let us prove that of the bike lock opens, then the last digits inserted were $\text{ABA}$.

• Consider an execution that opens the lock, i.e. that ends in state 4.

• As in 4 the formula $\text{pred}_3$ holds, the execution should end with 34.

• But in state 3 the formula $\text{pred}_2$ holds. Therefore the execution should end with 234.

• In state 2 and in state 4 the formula $\text{P}_A$ holds, and in state 3 the formula $\text{P}_B$ holds. Therefore the last three digits inserted should be: $\text{ABA}$.
Defining Models

- Kripke Structure

\[ K = \langle S, P, R, L, s_0 \rangle \]
- \( S \): the set of possible global states
- \( P \): a non-empty set of atomic propositions \( \{p_1, \ldots, p_k\} \) which express atomic properties of the global states, e.g., being an initial state, being an accepting state, or that a particular variable has a special value.
- \( R \subseteq S \times S \): a transition relation s.t. \( R(s, s') \) if \( s \) to \( s' \) is a possible atomic transition
- \( L: S \rightarrow 2^P \): a labeling function which defines which propositions hold in which states.
- \( s_0 \in S \): the initial state

- Model checking: A **model checker** checks whether a system, interpreted as an automaton, is a (Kripke) **model** of a property expressed as a **temporal logic formula**.

\[ K \models \varphi \]
The digicode automaton

- $S = \{1,2,3,4\}$
- $P = \{P_A, P_B, P_C, \text{pred}_1, \text{pred}_2, \text{pred}_3\}$
- $R = \{(1, A, 2), (1, B, 1), (1, C, 1), (2, A, 2), (2, B, 3), (2, C, 1), (3, A, 4), (3, B, 1), (3, C, 1)\}$
- $L = \{1 \mapsto \emptyset, 2 \mapsto \{P_A\}, 3 \mapsto \{P_B, \text{pred}_2\}, 4 \mapsto \{P_A, \text{pred}_3\}\}$
- $s_0 = 1$
Mutual Exclusion Example

- Two process mutual exclusive with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available (S₀) or taken (S₁)
- Initially both processes are in the Non-critical state and the semaphore is available --- N₁ N₂ S₀

\[
\begin{align*}
N₁ & \rightarrow T₁ \\
T₁ \land S₀ & \rightarrow C₁ \land S₁ \\
C₁ & \rightarrow N₁ \land S₀ \\
N₂ & \rightarrow T₂ \\
T₂ \land S₀ & \rightarrow C₂ \land S₁ \\
C₂ & \rightarrow N₂ \land S₀
\end{align*}
\]
Mutual Exclusion Example

- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1 N_2 S_0$

\[
\begin{align*}
N_1 & \rightarrow T_1 \\
T_1 \land S_0 & \rightarrow C_1 \land S_1 \\
C_1 & \rightarrow N_1 \land S_0 \\
N_2 & \rightarrow T_2 \\
T_2 \land S_0 & \rightarrow C_2 \land S_1 \\
C_2 & \rightarrow N_2 \land S_0
\end{align*}
\]
Mutual Exclusion Example

No matter where you are, there is always a way to get to the initial state.

\[ K \models AG \ EF (N_1 \land N_2 \land S_0) \]

Kripke structure  CTL (Computation Tree Logic)
Mutual Exclusion Example

Model
(System Requirements)

Specification
(System Property)

K ⊨ AG EF (N₁ ∧ N₂ ∧ S₀)

Model Checker
M ⊨ φ

Answer: Yes
Mutual Exclusion Example

Answer: Yes
A Proof: *For All* possible behaviors
Mutual Exclusion Example
Mutual Exclusion Example
Mutual Exclusion Example
Specification – Desirable Property

No matter where you are there is no way to get to the initial state

\[ K \models AG EF (N_1 \land N_2 \land S_0) \]
Mutual Exclusion Example

Answer: No
Counterexample
Printer Monitor Example

W = wait
P = print
R = rest

beg_A, end_A, req_A, beg_B, end_B, req_B

Printer Monitor Example
Printer Monitor Example

Desired properties

• In every execution, each state in which $P_A$ holds is preceded by a state in which $W_A$ holds
  – Easy to verify!

• In every execution every state in which $W_A$ is followed (sooner or later) by a state in which $P_A$ holds.
  – This property does not hold! And the model checker will produce a counterexample.
THE PRINTER MANAGER IS NOT FAIR

Counterexample: 0 1 3 4 1 3 4 1 3 4 1 3 4 1 3 4 1 3 4 ...
Automata with variables

• When we model a system we would like to represent also variables

• Program = Control + Data
  – The pair <state, transition> represents control
  – Variables represent data

• Example: In the digicode example, if we want to limit the number of attempts (max 3 errors), we need a counter that take count of the errors.
Interaction automaton - variables

• The automaton interacts with the variables in two ways:
  – **Assignment**: a transition may modify one or more variables
  – **Guard**: a transition may be constrained by the status of the variables
Dealing with variables and control stms

```c
int ctr;

if ctr<3
  B,C
  ctr++

if ctr<3
  A
  ctr++

if ctr<3
  B,C
  ctr++

if ctr=3
  B,C
  ctr++

if ctr=3
  A
  ctr++

if ctr=3
  A,C
  ctr++

if ctr=3
  B,C
  ctr++

if ctr=3
  B,C
  ctr++

err
```
Unfolding

- Automata with variables can be expressed in automata state graph where only state transactions appear.
- In this case we speak about a “transition system”.
- The states of an unfolded automaton are called global states.
Unfolding
Syncronization: an elevator
The doors at the different floors

Diagram:

- States: Closed, Open
- Transitions: 
  - Closed → Open: ?close
  - Open → Closed: ?open_i
  - Closed → Closed: ?close_i
  - Open → Open: ?open_i
The controller

Diagram:

- **free2**
  - !close_2
  - !open_2

- **free1**
  - !close_1
  - !open_1

- **free0**
  - !close_0
  - !open_0

- **on0**
  - !down

- **on1**
  - !down

- **on2**
  - !down

- **2->0**
- **0->2**
The resulting automaton

- The system is represented by the product of 5 automata (3 doors, the elevator, the controller)
- The constraints are represented by conditions on the transactions:

\[
\text{Sync} = \{(\text{?open}_0, -, -, -, \lnot \text{open}_0), \quad (\text{?close}_0, -, -, -, \lnot \text{close}_0), \\
(\text{-}, \text{?open}_1, -, -, \lnot \text{open}_1), \quad (\text{-}, \text{?close}_1, -, -, \lnot \text{close}_1), \\
(\text{-}, \text{-}, \text{?open}_2, -, \lnot \text{open}_2), \quad (\text{-}, \text{-}, \text{?close}_2, -, \lnot \text{close}_2), \\
(\text{-}, \text{-}, \text{-}, \text{?down}, \lnot \text{down}), \quad (\text{-}, \text{-}, \text{-}, \text{?up}, \lnot \text{up}) \}
\]
Desired properties

• The door at a given floor does not open if the elevator is at a different floor.
• The elevator does not move if one door is still open