5. Model Checking
Verification?

“Testing can only show the presence of errors never their absence”
(Edsgar Dijkstra)
Model Checking

- Proof of $M \models \varphi$
- Question: „Is the system a model of the formula?“
- Model Checking algorithm searches the whole state space of the system
  → State space needs to be finite
Model Checking

- What is Model Checking? Two answers:
  - … an automated proof method
  - … a precise system analysis tool

- A rigorous proof that the program does what you say it should do

- This requires:
  - Formal semantics: Kripke structure
  - Formal specification: Temporal logic
  - Method: Manual proof (up to 1981); today model checking!

"Reused" slides (partially modified/combined) from:
[Dwyer2002s] Matthew Dwyer. Kansas State University, USA Software Model Checking Tutorial
FSE’02 – Charleston, South Carolina – Nov. 19, 2002
Input-Output Patterns

- Reactive systems:
  - Parallelism, distribution
  - Interaction with their environment, no termination, hence Hoare Style proofs do not work
  - High complexity, safety critical
Model Checking

UML-Diagram (e.g. StateChart)

Kripke-structure

Model Checker

Code Generation & Test

Specification

It is always true...
It is never true...

Temporal logic formula (e.g. CTL, LTL, ...)

Counter example
**Kripke Structure**

- Defines a state based transition graph, which specifies the behavior of a reactive system

Kripke Structure = (AP, S, L, T, S₀)

- AP: Set of atomic propositions
- S: finite set of states
- L: S → ℙ(AP), labels
- T ⊆ S x S, transitions, is left-total (every node is source node for at least one transition)
- S₀: set of initial states

\[
\begin{align*}
\text{AP} &= \{p_0, p_1, p_2\} \\
\text{S} &= \{s_0, s_1, s_2, s_3, s_4\} \\
\text{L} &= \{s_0 \rightarrow \{p_0\}, s_1 \rightarrow \{p_1\}, \{s_2\} \rightarrow \{p_0\}, s_3 \rightarrow \{p_2\}, s_4 \rightarrow \{p_1, p_2\}\} \\
\text{T} &= \{(s_0, s_1), (s_1, s_0), (s_1, s_2), (s_1, s_3), (s_2, s_3), (s_3, s_0), (s_3, s_4), (s_4, s_4)\} \\
\text{S₀} &= \{s_0\}
\end{align*}
\]
Kripke Structure → Path Tree

\[ s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_4 \]

\[ p_0 \rightarrow p_1 \rightarrow p_2 \]

\[ \ldots \]

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \]

\[ p_0 \rightarrow p_1 \rightarrow p_2 \]

\[ \ldots \]
Microwave Example

AP := \{start, close, error, heat\}
Microwave Example

- Properties:
  - There is a state where MW heats
  - The MW shall not heat if an error occurred
  - There is a state directly following the initial state where the door is closed

AP := \{start, close, error, heat\}
Computation Tree Logic (CTL)

- CTL-Formula consist of:
  - Atomic propositions (predicates on states)
  - Boolean connectives $\land$, $\lor$, $\neg$, $\Rightarrow$
  - Additional CTL-operators of the form:
    
    path quantifier + temporal operator

- Path Quantifier:
  - **A**: for all paths holds…
  - **E**: there is a path for which holds…

- Temporal Operator:
  - **X**: Next
  - **F**: Future
  - **G**: Global
  - **U**: Until
Computation Tree Logic (CTL)

- Valid CTL-formulae are defined recursively as follows:
  - Every atomic proposition \((p, q, \ldots)\) is a valid CTL-Formula.
  - Let \(\phi_1, \phi_2\) be CTL-formulae then the following formulae are valid CTL-formulae:
    - \(\neg \phi_1\)
    - \(\phi_1 \land \phi_2\)
    - \(\phi_1 \lor \phi_2\)
    - \(\phi_1 \Rightarrow \phi_2\)
    - \(\text{EX} \phi_1\)
    - \(\text{EF} \phi_1\)
    - \(\text{AG} \phi_1\)
    - \(\text{EG} \phi_1\)
    - \(E[ \phi_1 \text{ U } \phi_2 ]\)
    - \(A[ \phi_1 \text{ U } \phi_2 ]\)
    - \(\neg \phi_1\)
    - \(\phi_1 \land \phi_2\)
    - \(\phi_1 \lor \phi_2\)
    - \(\phi_1 \Rightarrow \phi_2\)
    - \(\text{AX} \phi_1\)
    - \(\text{AF} \phi_1\)
    - \(\text{AG} \phi_1\)
    - \(\text{EG} \phi_1\)
    - \(\text{EF} \neg \phi_1\)
    - \(\text{EX} \phi_1\)

- Examples: \(p, \neg p, p \land q, \text{EF} p, \text{AG} q, \neg \text{AG} q, \text{AG} (\text{EF} \neg p), (\text{EX} p) \lor (\text{AG} q), \text{A}[ (\text{EF} p) \text{ U } q ]\)
CTL-Operators Examples

„There is a path where p holds in the next state.“

EX p

„There is a path where p holds Until q holds.“

E[pUq]

„For all paths p holds at some point in the Future.“

AF p

„For all paths p holds Globally (in every state).“

AG p

CTL: A path quantifier in front of every temporal operator
Microwave Example

- Properties:
  - There is a state where MW heats
    - $\text{EF} \ (\text{heat})$
  - The MW shall not heat if an error occurred
    - $\text{AG} \neg (\text{error} \land \text{heat})$
  - There is a state directly following the initial state where the door is closed
    - $\text{EX} \ (\text{close})$

$\text{AP} := \{\text{start, close, error, heat}\}$
State Explosion Problem

- Problem: number of states of a realistic system usually huge
- The number of states of a system is exponential in its number of variables
  → Construction of the whole state space not always feasible (too little system memory)

- Solution: try to avoid explicit construction of state space (Later in this course)
Software Model Checking: Reality Check

Specifications are partial
- They do not define complete functional correctness
- Focus on crucial properties

Cost of checking is enormous
- Must approximate
- What kinds of approximation are useful?

[Dwyer2002s]
Software Model Checking - Approximation

Model Checker

Software model

Specification

\( \square (\Phi \rightarrow \diamond \Omega) \)

Approximation

No false positives

OK

No false negatives

Error trace

Line 5: ...
Line 12: ...
Line 15: ...
Line 21: ...
Line 25: ...
Line 27: ...
...
Line 41: ...
Line 47: ...

[Dwyer2002s]
Model Construction Problem

Semantic gap:
- Programming Languages with methods, inheritance, dynamic creation, exceptions, etc.
- Model Description Languages are only automata

[Dwyer2002s]
Output Interpretation Problem

- Raw error trace may be 1000’s of steps long
- Must map line listing onto model description
- Mapping to source is made difficult by
  - Semantic gap & clever encodings of complex features
  - multiple optimizations and transformations
Why is model-checking software difficult?

- Problems using existing checkers:
  - Model construction
  - Property specification
  - State explosion
  - Output interpretation

\[ \square (\Phi \rightarrow \Diamond \Omega) \]

Finite-state model

Model Checker

OK

or

Error trace

[Dwyer2002s]
Summary

- Instead of a „complete“ specification use only one that consists of relevant properties (e.g., for safety)
- Usually only restricted notions for formal models
  - Finite automata (or similar restricted models)
- Often restricted notions for formal properties
  - Propositional logic
  - Temporal logic (Model-Checking)

- Benefit:
  - Counterexample when a property is not fulfilled
- Limitations:
  - Not feasible for too large models (state explosion)
  - Not feasible for too complex formal properties
5.1 CTL Model Checking
Microwave Example

Properties:

- There is a state where MW heats
  - $\text{EF (heat)}$
- The MW shall not heat if an error occurred
  - $\text{AG } \neg(\text{error } \land \text{heat})$
- There is a state directly following the initial state where the door is closed
  - $\text{EX (close)}$

$\text{AP} := \{\text{start, close, error, heat}\}$
CTL-Formula Reduction

- By EX, EG and EU all other properties can be expressed, as e.g. in
  - AX f = ¬EX (¬f)
  - EF f = E (true U f)
  - AG f = ¬EF (¬f)
  - AF f = ¬EG (¬f)
  - A (f U g) = ¬E (¬g U (¬f ∧ ¬g)) ∧ ¬EG ¬g
    = ¬(E (¬g U (¬f ∨ g)) ∨ EG ¬g)

- In the following only EX, EG, EU, ¬ and ∨
Model Checking Algorithm (1/2)

- Globally defined Kripke-Structure,
- label(s) is the set of all formulas which hold in a state s

- Iterative algorithm: In the i-th run check all sub-formulae of height i in the syntax tree of the formula
  - ensures that no formula is examined before its sub-formulae are examined

- Example: \( \text{EX} (p \lor \text{EG} \neg q) \)

```plaintext
function verify (Formula f) : boolean;
    for i = 0 .. h do
        for all sub-formulae g of f of height i do
            check (g);
        end for all;
    end for;
    return \( \forall s \in S_0: f \in \text{label}(s) \)
end verify;
```

(h is height of the syntax tree of f)
function check (Formula f);
    switch
    {
    case f ∈ AP  checkAP(f)  // label states with their atomic propositions
    case f = ¬g    checkNegation(g)  // check negations
    case f = g ∨ h checkDisjunction(g, h)  // check disjunction
    case f = EX g checkEX(g)
    case f = EG g checkEG(g)
    case f = E[gU h] checkEU(g, h)
    end switch
    end check;
**checkAP**

- Function checkAP(g) marks a state s with the atomic proposition g if g holds in s

```plaintext
function checkAP(g)
    for all s ∈ S do
        if g ∈ label(s) then
            label(s) := label(s) ∪ {g}
        end labelStates;
end
```

![Diagram showing a graph transformation](image)
function check (Formula f);
    switch
    
    case f ∈ AP checkAP(f)  // label states with their atomic propositions

    case f = ¬g checkNegation(g) // check negations
    case f = g \lor h checkDisjunction(g, h) // check disjunction
    case f = EX g checkEX(g)
    case f = EG g checkEG(g)
    case f = E[gU h] checkEU(g, h)
    
end switch
end check;
Function checkNegation(g) marks a state s with $\neg g$ if the sub-formula g does not hold in s

function checkNegation(g)
    for all $s \in S$ do
        if $g \notin \text{label}(s)$ then
            $\text{label}(s) := \text{label}(s) \cup \{\neg g\}$
    end labelStates;

Diagram:

- Initial state: $g$ with transitions $g \rightarrow f$, $g \rightarrow h$, $g \rightarrow g$
- After applying checkNegation with $g$: $g$ with transitions $g \rightarrow f$, $g \rightarrow h$, $g \rightarrow g$, and $g \rightarrow \neg g$
function check (Formula f);
    switch
    
    case f ∈ AP checkAP(f)  // label states with their atomic propositions
    case f = ¬g checkNegation(g)  // check negations
    case f = g ∨ h checkDisjunction(g, h)  // check disjunction
    case f = EX g checkEX(g)
    case f = EG g checkEG(g)
    case f = E[gU h] checkEU(g, h)
    
    end switch
end check;
**checkDisjunction**

- Function `checkDisjunction(g, h)` marks all states for which `g` or `h` holds

```plaintext
function checkDisjunction(g, h)
    for all s ∈ S do
        if g ∈ label(s) or h ∈ label(s) then
            label(s) := label(s) ∪ {g ∨ h}
        end if
    end for
end function
```
Model Checking Algorithm
(2/2)

function check (Formula f);
switch
  case f ∈ AP   checkAP(f)  // label states with their atomic propositions
  case f = ¬g   checkNegation(g)  // check negations
  case f = g ∨ h checkDisjunction(g, h)  // check disjunction
  case f = EX g  checkEX(g)
  case f = EG g   checkEG(g)
  case f = E[gUh] checkEU(g, h)
end switch
end check;
CheckEX

- Marks all states for which \( \text{EX}(g) \) holds
- Is there a path for which \( g \) holds in the next state?

**function** `checkEX(g)`

\[
S_1 := \{ s \mid g \in \text{label}(s) \}
\]

**for all** \((s_1, s_2) \in T, s_2 \in S_1\)**

\[
\text{label}(s_1) := \text{label}(s_1) \cup \{ \text{EX} \ g \}
\]

**end checkEX;**

- Remark: \( T \subseteq S \times S \) is left-total
Model Checking Algorithm (2/2)

function check (Formula f);
  switch
  case f ∈ AP  checkAP(f)  // label states with their atomic propositions
  case f = ¬g  checkNegation(g)  // check negations
  case f = g ∨ h  checkDisjunction(g, h)  // check disjunction
  case f = EX g  checkEX(g)
  case f = EG g  checkEG(g)
  case f = E[gUh]  checkEU(g, h)
  end switch
end check;
**Strongly Connected Component**

- **Given:** directed Graph G
- **Strongly Connected Component C:**
  - Subgraph of G
  - Any node of C is reachable by a (directed) path from any other node in C
- **Non trivial Strongly Connected Component**
  - C is a Strongly Connected Component
  - C has either more than one node or one node with a self referencing edge
Example

Trivial SCC

Non trivial SCC

Non trivial SCC
CheckEG: Example

- Marks all states for which EG (f) holds
- Is there a path for which f always holds?
CheckEG: Example

- M0 = \{S3, S4, S5\}
- M1 = \{S3, S4, S5, S2\}
- M2 = \{S3, S4, S5, S2, S1\}
function checkEG(f)
    \( S_1 := \{ s \mid f \in \text{label}(s) \} \)
    \( \text{SCC} := \{ C \mid \text{nontrivial SCC of } S_1 \} \)
    \( S_2 := \bigcup_{c \in \text{SCC}} \{ s \mid s \in C \} \)
    for all \( s \in S_2 \) do \( \text{label}(s) := \text{label}(s) \cup \{ \text{EG } f \} \) end for all
    while \( S_2 \neq \emptyset \) do
        choose \( s \in S_2 \)
        \( S_2 := S_2 \setminus \{ s \} \)
        for all \( (s', s) \in T, s' \in S_1 \) do
            if \( \text{EG } f \notin \text{label}(s') \) then
                \( \text{label}(s') := \text{label}(s') \cup \{ \text{EG } f \} \)
                \( S_2 := S_2 \cup \{ s' \} \)
            end if
        end for all
    end while
end checkEG
function check (Formula f);
    switch
        case f ∈ AP    checkAP(f)        // label states with their
                                             // atomic propositions
        case f = ¬g    checkNegation(g)   // check negations
        case f = g ∨ h checkDisjunction(g, h) // check disjunction
        case f = EX g  checkEX(g)
        case f = EG g  checkEG(g)
        case f = E[gUh] checkEU(g, h)
    end switch
end check;
CheckEU

- Marks all states for which $E[fUg]$ holds
- CheckEU($f,g$): is there a path where $f$ holds until $g$ holds

- $M_0 = \{S4, S5\}$
- $M_1 = \{S4, S5, S2\}$
- $M_2 = \{S4, S5, S2, S1\}$
function CheckEU(f, g)
    \[ S_1 := \{ s | g \in \text{label}(s) \} \]
    for all \( s \in S_1 \) do label(s) := label(s) \cup \{ E[fUg] \} end for all
    while \( S_1 \neq \emptyset \)
        choose \( s \in S_1 \)
        \( S_1 := S_1 \setminus \{ s \} \)
        for all \( (s', s) \in T \) do
            if \( E[fUg] \notin \text{label}(s') \) and \( f \in \text{label}(s') \) then
                label(s') := label(s') \cup \{ E[fUg] \}
                \( S_1 := S_1 \cup \{ s' \} \)
            end if
        end for all
    end while
end checkEU