Design of Reactive Systems
Summer 2002

Prof. Dr. Stefan Leue
Institute for Computer Science
Albert-Ludwigs-Universität Freiburg

leue@uni-freiburg.de

Copyright © Stefan Leue 2002
Automata-based Model Checking

Part 6
Automata Models and Logics

♦ Automata
  ‣ operational requirements specifications and architectural design specification for reactive systems are often captured in automata-based formalisms
    – SDL
      • ITU-T Recommendation Z.100
      • (H)CEFSM-Modell with arbitrary data domains (ASN.1 abstract data type specification language) and unbounded message buffers
    – UML for Real-Time (UML-RT)
      • HCEFSM

♦ Logics
  ‣ to formalize more abstract, descriptive specifications we need a more abstract notation
    – use of logic to characterize abstract requirements
Example: Requirements Validation

- Requirements elicitation
- Requirements analysis and negotiation
- Requirements documentation and specification
- Negotiated and validated requirements

\[ M \models L \]

"model checking"

Logic specification

Customer or user requirements (abstract)
Requirements Validation

L (abstract requirements)

M (operational requirements model)
Further Applications of Model Checking

- Does the design implement the requirements specification?
- Does an implementation satisfy its specification (design / requirements)?
  - "software model checking"

System design → Requirements → Design → Implementation → Integration → Maintenance
Model Checking Principle

♦ Example

› mutual exclusion
  – two processes, i = 1, 2
  – state - transition model
    • global state space of mutual exclusion algorithms, states S1, .., S9
    • transitions represent spontaneous, atomic actions on the global state space (interleaving semantics approach)
  – state propositions
    • n_i: process i is not in the critical section
    • t_i: process i tries to get access to the critical section
    • c_i: process i is in the critical section
Model Checking Principle

♦ A Mutual Exclusion System
Safety requirement
- There are never going to be two processes in the critical section at the same time

Formalization
- \( \neg (c1 \land c2) \)
Safety requirement

- validation principle: visit every state and check satisfaction of property
- if all states satisfy the property, then it is impossible to find an execution prefix that could be continued such that the property would be violated
*Liveness Property*

- *it is true in every state that every process will eventually enter the critical section*
- formalization
  - ♦ (c1 ∨ c2)
Liveness Property

- validation principle
  - visit every state and check, whether a cycle violating the condition can be found
  - i.e., not all infinite extensions of all finite prefixes satisfy the property
**Response**
- Once a process attempts to get access to the critical section, then it will eventually be granted access.

**Formalisation**
- \((t_1 \rightarrow \Diamond c_1) \land (t_2 \rightarrow \Diamond c_2)\)
Model Checking Principle

- Search Strategies
  - safety: depth-first-search
  - liveness: nested DFS

- General Strategies?
  - Model checking for arbitrary LTL formulae
Temporal Logic and Automata

♦ Relationship of Büchi-Automata and LTL
  ‣ LTL corresponds to counter-free Büchi-automata and *-free $\omega$-regular expressions
  ‣ Büchi-automata are strictly more expressive than LTL formulae
    – comparison according to [Wolper]:

\[
\begin{array}{c|c|c}
\text{ETL} & \iff & \text{Not counter-free Büchi-automata} \\
\text{LTL} & \iff & \approx \omega\text{-regular languages} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{Counter-free Büchi-automata} & \approx & \text{*-free } \omega\text{-regular languages} \\
\end{array}
\]
Temporal Logic and Automata

♦ Büchi-Automata vs. LTL

example:

even(p): Proposition p is true in every even numbered state of a state sequence

Büchi:

\[
\begin{array}{c}
\text{S1} \\
\text{p} \\
\text{S0}
\end{array}
\]

ω-reg: \((\Sigma p)^\omega\)

LTL (intuitive argument):

\[\phi: p \land (p \rightarrow \bigcirc \neg p) \land (\neg p \rightarrow \bigcirc p)\]

cannot fully characterize this property

let \(\sigma = \underbrace{p \mid p \mid p \mid \cdots}_{\text{even(p)}}\)

then \(\sigma \models \text{even(p)}\) and \(\sigma \not\models \phi\)

conclusion:

− properties that require repeated counting up to a constant n cannot be expressed in LTL
− Büchi automata can count modulo a constant n
− for every LTL formula there is an expressively equivalent Büchi-automaton
− for proofs see [Wolper]
Temporal Logic and Automata

♦ LTL Formulae and Equivalent Automata

\( \neg(c_1 \land c_2) \)

\( \diamondsuit (c_1 \lor c_2) \)
Temporal Logic and Automata

♦ LTL Formulae and Equivalent Automata

\[(p \rightarrow \diamond q) \iff (\neg p \lor \diamond q)\]

\[\diamond (p \land \neg q)\]
Temporal Logic and Automata

♦ LTL Formulae and Equivalent Automata

\[
(p \rightarrow (r \land q))
\]

\[
(p \rightarrow (r \lor q))
\]

\[
\lozenge(p \land (\neg r \lor q))
\]
Automata-based Model Checking

♦ Approach

- let S a specification
- let M a model
- let \( L(S) \) and \( L(M) \) the languages accepted by S and M, respectively
- M satisfies the specification S if
  \[ L(M) \subseteq L(S) \]
Automata-based Model Checking

♦ Application to Requirements Validation

\[ L(M) \subseteq L(S) \]
Automata-based Model Checking

♦ **Computing** $L(M) \subseteq L(S)$
  - let $\Sigma^\omega - L(S) = \text{comp}(L(S))$
  - then
    $$L(M) \subseteq L(S) \iff L(M) \cap \text{comp}((L(S)) = \emptyset$$
  - Büchi automata are closed under complement and intersection, i.e.,
    - there is always a B.A. representing $\text{comp}((L(S))$, and
    - there is always a B.A. representing $L(A) \cap \text{comp}((L(S))$

♦ **Computing** $L(M) \cap \text{comp}((L(S)) = \emptyset$
  - implementation variants
    - direct complementation of $S$: non-trivial operation (c.f. [Sistla, Vardi and Wolper])
    - direct specification of $\text{comp}(S)$
    - specify LTL formula $\varphi$, obtain $\neg \varphi$, translate $\neg \varphi$ into an equivalent B.A.
Automata-based Model Checking

♦ Existence of Counterexample
  
  ‣ \( L(M) \cap \text{comp}((L(S)) = \emptyset \)
    
    \( \Rightarrow \) A satisfies S
  
  ‣ \( L(M) \cap \text{comp}((L(S)) = C \neq \emptyset \)
    
    \( \Rightarrow \) C is counterexample for the non-satisfaction of S by M
  
  ‣ it can be shown that any word in C can be represented by an \( \omega \)-regular expression of the form \( uv^\omega \) where u and v are finite state of event sequences
Automata-based Model Checking

♦ Construction of the Intersection of Büchi Automata

- let
  \[ M^1 = (Q^1, q_0^1, A, \delta^1, F^1) \] and
  \[ M^2 = (Q^2, q_0^2, A, \delta^2, F^2) \]
  two Büchi automata

- the Büchi automaton accepting \( L(M^1) \cap L(M^2) \) can be defined as
  \[ M^1 \cap M^2 = (Q^1 \times Q^2 \times \{0, 1, 2\}, (q_0^1, q_0^2, 0), A, \delta, Q^1 \times Q^2 \times \{2\}) \]
  such that \((<r_i, q_j, x>, a, <r_k, q_n, y>) \in \delta\) iff all of the following conditions hold:

  1. \((r_i, a, r_k) \in \delta^1\) and \((q_j, a, q_n) \in \delta^2\) (the transitions of the intersection automaton agree with the transitions of the operand automata)

  2. The third component of the state tuples can be computed as follows:
     * if \(x=0\) and \(r_k \in F^1\), then \(y = 1\)
     * if \(x=1\) and \(q_n \in F^2\), then \(y = 2\)
     * if \(x=2\) then \(y = 0\)
     * else, \(y=x\)
Automata-based Model Checking

Example (from [Clarke, Grumberg and Peled])

M₁

\[ r₁ \quad a \quad b \quad r₂ \]

M₂

\[ q₁ \quad b \quad a \quad q₂ \]

M₁ ∩ M₂

\[ r₁,q₁,0 \quad a \quad r₂,q₁,0 \]

\[ r₁,q₂,1 \quad a \quad r₂,q₁,0 \]

\[ r₂,q₁,0 \quad b \quad r₁,q₂,0 \]

\[ r₂,q₁,2 \quad b \quad r₁,q₂,0 \]
Automata-based Model Checking

Construction of the Intersection of Büchi Automata

- third component in state ensures that acceptance states of both operand automata $M^1$ and $M^2$ occur infinitely often in an accepting cycle
- a definition of $F = F^1 \times F^2$ would be insufficient since accepting cycles of only one of the operand automata would be accepted in the intersection
- third component of state:
  - from 0 to 1 when an accepting state of the first automaton is reached
  - from 1 to 2 when additionally an accepting state of the second automaton is reached (accepting state of the intersection)
  - from 2 to 0 in the next state
Automata-based Model Checking

♦ Emptyness Check for Büchi Automata

- let $M = (Q, q_0, A, \delta, F)$ a Büchi automaton
- let $\rho$ a run on $\sigma$ such that $\sigma$ is being accepted by $M$
- $\rho$ contains infinitely many accepting states from $F$
- since $Q$ is finite, there is a suffix $\rho'$ of $\rho$ such that every state in $\rho'$ occurs infinitely often
- Every state in $\rho'$ is reachable from every other state in $\rho'$
  - i.e., the states in $\rho'$ form part of a strongly connected component of $M$ that is reachable from $q_0$
  - i.e., the problem $L(M) = \emptyset$ is equivalent to finding a strongly connected component within the state graph of $M$
- $\rightarrow$ Tarjan’s depth first search, (DFS)
  - linear time in the size of $M$
  - derivation of counterexample
    - the members of the SCC form the infinitely often repeated suffix
    - the reachability path for the component from $q_0$ forms prefix
- more efficient for practical problems: nested DFS
Automata-based Model Checking

♦ Nested DFS

```
procedure emptiness
dfs(q₀);
terminate(false)
end procedure

procedure dfs1(q)
local q’;
hash(q);
for all successors q’ of q do
  if q’ not in hashtable then dfs1(q’)
  end if;
if q ∈ F then dfs2(q)
end if;
end do;
end procedure

procedure dfs2(q)
local q’;
flaq(q);
for all successors q’ of q do
  if q’ on dfs1-stack then terminate(true)
  else if q’ not flagged then dfs2(q’)
  end if;
end do;
end procedure
```
Automata-based Model Checking

♦ Nested DFS

♦ derivation of counterexample for \( \text{terminate(true)} \)
  – \( \text{dfs2}(s_1) \)
  – \( s_2 \) is found by dfs2 on dfs1-stack
  – counterexample construction:
Automata-based Model Checking

♦ Synchronous Product ([Holzmann 95])

    ‣ idea
      – let M a B.A. corresponding to the transition system that is to be validated
      – let L a B.A. corresponding to the property that is to be validated
        • transitions from L are labeled with propositional expressions referring to state variables of M
        • perform a matching between the states of M and the transitions of L

\[
\begin{align*}
\text{M: } s'_0, \ldots, s'_i=v, \ldots \\
\text{L: } s_0, \ldots, s_i=x, s_{i+1}=y, \ldots
\end{align*}
\]
Automata-based Model Checking

♦ Synchronous Product ([Holzmann 95])

idea

– goal: generate all accepting runs of L that are also accepting runs of M
– a run of L matches a run of M, if all states in L’s run match all states of M’s run
– An accepting run on L which matches a run on M describes an execution sequence corresponding to the (undesired) property expressed by L
– Absence of an accepting run of L, for which there is an accepting run on M, implies that the (undesired) property does not hold
Automata-based Model Checking

♦ Synchronous Product ([Holzmann 95])

Example

\[ at_i \]

\[ \neg at_i \]

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]

\[ M \]

\[ L \]

\[ \text{wahr} \]

\[ y_1 \]

\[ y_2 \]

\[ \neg at_i \]

\[ \neg at_i \]

\[ \neg at_i \]

\[ (x_1, y_1) \]

\[ (x_2, y_1) \]

\[ (x_1, y_2) \]

\[ (x_3, y_1) \]

\[ (x_3, y_2) \]

\[ P \]

\[ \Diamond at_i \]

\[ \neg at_i \]

Negation
Automata-based Model Checking

♦ Synchronous Product ([Holzmann 95])
  ‣ Example
    – Any run of the product automaton $P$ corresponds to a run on $M$ as much as to a run on $L$
    – Any accepting run on $P$ hence corresponds to a run on $M$ and an accepting run on $L$, which shows that the intersection of the languages accepted by $M$ and $L$ is not empty
Automata-based Model Checking

♦ Synchronous Product ([Holzmann 95])
    Example
    – the property expressed by L holds for M since the synchronous product P has an accepting cycle
    ‧ detected through nested DFS on P
    – counterexample: \((x_1, y_1), (x_3, y_1), (x_3, y_2), (x_3, y_2)\)
Automata-based Model Checking

♦ Synchronous Product ([Holzmann 95])

Example 2

No accepting run of P, hence the (undesired) property is not satisfied.
Bibliographic References